New Solution Schemes in Constrained Redundancy Optimizatin

by

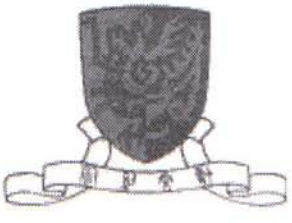
LAM Ngok

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy

in

Systems Engineering and Engineering Management

1999



摘要

在我們的研究中，我們探討了"有約束冗餘最優化問題" 的解題方法。這類型題目是屬於整數非線性規劃範疇中的問題。這類題目的目標函數是非線性並且不可分。因此採用動態規劃般的高效率解題方法來解決這類題目是十分困難的。所以我們提出了一些新的方法來更有效及更快速地處理這類型題目。

近期研究顯示，平行-串聯線路的資源分配最優解有一個十分有趣的特性，這特性就是在可行情況下，額外的同類部件應該放置在可靠性低的部件處。我們研究了這有趣現象，并發現了一個放置額外部件的系統上限。這上限是隨部件分配不同而改變。當部件分配的差超越上限，而我們又能找到一些可行但卻沒超越分配上限的系統資源分配時，本來那個超越了上限的資源分配方案就可以放棄不理。我們同時也嘗試把這結果擴展到平行-串聯可化簡型線路中，並發現了一些類似的特性。

我們在論文的後半部展示了針對"有約束冗餘最優化問題" 中一般複雜系統的一個新的解題方法。這種方法能把問題的目標函數凹化，並轉換成可分離的線性輔助問題，再從這輔助問題中找出本來問題的最優解。我們將會用這新方法來展示一些解題例子。

To my parents

Acknowledgements

I would like to thank all the people that have offered their generous help to me over the last two years.

I particularly would like to thank my supervisor, Professor Li Duan for his kind and patient guidance, extremely valuable suggestions and advices throughout the last two years. Without Professor Li's insights and his endless encouragement, this paper may never be completed.

Besides, I would also like to thank Chu Sir, Lau Kwok Kin, Yung Hoi Wing, Francis Lam, Law Muk Yun, Mun Wo Di, Chan Kun Chung, Ng King Kwok, Leung Wai Kwong, Yu Kwok Leung, Chan Kin Pong, Jason Choi, Kaiser So, Woo Yuk Foo Yip Chung Fat, Matt Chan for their helps and their encouragements in the last two years.

Abstract

We investigate into the constrained redundancy optimization problems which belong to the category of nonlinear integer programming problems in our research. The objective functions of this type of problems always bear nonlinear problem structures that are not separable. They are therefore hard, if not impossible, to be solved directly by common efficient methodologies like the dynamic programming methodology. We propose some new methods to deal with the problems in a more efficient and effective way.

Recent research discovered that optimum resource allocation of series-parallel reliability networks has an interesting characteristic such that the redundancies should be put to the subsystems in descending order of their reliabilities if this is possible. In other words, more redundancies should be allocated to the less reliable subsystems whenever feasible. We investigate into this interesting phenomenon, and further identify an upper bound that governs the optimal differences between the redundancy levels. Whenever a difference between the redundancy levels in a redundancy assignment exceeds the upper bound which is described by the equation we derived , this very redundancy assignment should then be ignored and precluded from further considerations if there exists a feasible redundancy assignment which has the difference smaller than the aforesaid upper bound. In addition to this, we

also extend the findings to the series-parallel reducible networks and identify some similar properties.

We also show in our later chapter a novel methodology developed by Li [1], which is capable of concavifying the unseparable constrained redundancy optimization problems and converting them into a separable auxiliary parametric problem that can be solved directly by the dynamic programming method. We will show some examples to illustrate how this new methodology works.

Contents

List of Figures

1 Introduction

1.1 Overview

Constrained redundancy optimization problems belong to the category of nonlinear integer programming problems. This type of problems are more difficult to be solved than the general nonlinear programming problems since their solutions must be integers. Thought many techniques have been proposed over the years, none seems to be superior over the other in solving large scale problems.

Current problem-solving algorithms can be divided into three broad categories, namely the approximation schemes, the heuristic schemes and the exact solution schemes. The approximation schemes like the geometric programming schemes [2] require the problems to be formulated as positive polynomials. They give approximated continuous solutions. These continuous solutions are required to be rounded to the closest integers which may not necessarily be the gobal optimal solutions. Heuristic schemes can generate solutions for the problems in a relatively short period of time, but gobal optimality of the solutions is not guaranteed. Exact solution schemes are the only schemes that are capable of generating gobal optimal solutions, but they are slower and may require special problem structures that are unavailable on most of the constrained redundancy optimzation problems. This is the

background of our research.

In order to deal with the constrained redundancy problems, we develop a new exact searching algorithm to attack the problems. A new approach by considering the relationship between component reliabilites and component redundancy levels is adopted. Computational results show that the new alogrithm can considerably reduce the search space for the constrained redundancy problems. We shall provide a detailed discussion on this in chapter 4 and chapter 5. Another novel successive solution scheme developed by Li[1] is also investigated in this thesis. This new solution scheme is capable of converting the unseparable problems into separable auxiliary problems and generating gobal optimal solutions for them, we tried this new method with two example problems and the results are listed in Chapter 6.

1.2 Organization outline

This thesis is organized as follows: Chapter 1 is an introduction section that gives a very brief introduction on our research. Chapter 2 presents the reader with some fundamental knowledges of the reliability studies. Chapter 3 is a literature review section which provides a review on the major literatures. Chapter 4 and 5 show the results achieved in our research on series-parallel networks and series-parallel reducible networks. We then have some dis-

cussions on the "Successive Solution Scheme For Constrained Redundancy Optimization In Reliability Networks" developed by Li [1] in chapter 6. Two example problems solved by this new solution scheme are also included at the end of Chapter 6. Chapter 7 is the conclusion section which gives a conclusion and some recommendations for possible further improvements. Chapter 8 is an appendix that contains the computational results and the programme codes we developed for the alogrithm we derived in Chapter 4.

2 Fundamentals of reliability theory

Reliability theory is concerned with determining the probability such that a system will function [3]. The "reliability" of a system is the probability of successful operation of the system for a specific period of time [4]. In our study, the main objective is to optimize the reliabilities of the general systems under certain constraints. Before going deep into our discussions, we need to present some simple concepts that are fundamental to our research.

2.1 State vector

For a system of n components, a variable x_i is defined in the following way :

$$x_i = \begin{cases} 1, \ if \ the \ i^{th} \ component \ is \ functioning. \\ 0, \ if \ the \ i^{th} \ component \ has \ failed. \end{cases}$$

The vector $X = (x_1, x_2, ..., x_n)$ is known as the state vector of the system. It indicates the status of the components of the system. (In our later discussions in Chapter 4 and Chapter 5, however, the X is also used to denote the redundancy assignments of a system, we will explicitly state this then)

2.2 Minimal path sets

Any system can be represented as either 1) parallel arrangements of series networks or 2) serial arrangements of parallel networks. We will first try to

represent the systems as parallel arrangements of series networks by representing the systems into collections of minimal path sets.

The minimal path sets are the sets that contain the minimal paths for the system to function. They are the minimal sets of components to ensure the functioning of the systems. Each minimal path sets contains exactly one minimal path to the system. For the network in Fig.1, the minimal path sets are {1,2,} and {1,3}.

The exhaustive collection of the minimal path sets of a system holds all the series networks that are conceptually in parallel with one another. With all these parallel paths together, one can form another system which is equivalent to the original system. To illustrate this, consider again the network in Fig.1. It consists of two minimal path sets {1,2} and {1,3}. It can be viewed as a system consisted of two parallel paths, one with the components 1, 2 connecting in series while the other one with components 1, 3 in series. The original system in Fig.1 can now be represented as the network in Fig.2, which is a parallel arrangement of serial networks. Please note that Fig.1 and Fig.2 are equivalent.

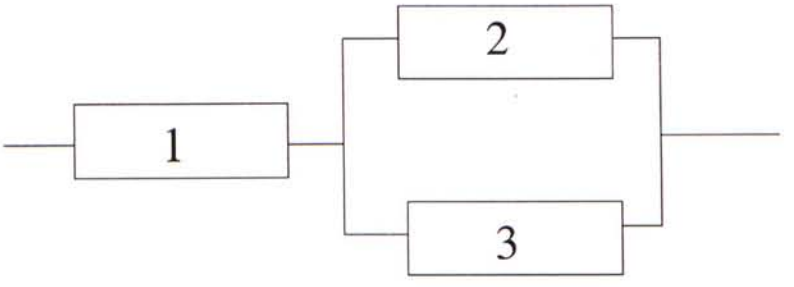


Figure 1: A network example to illustrate the minimal path sets

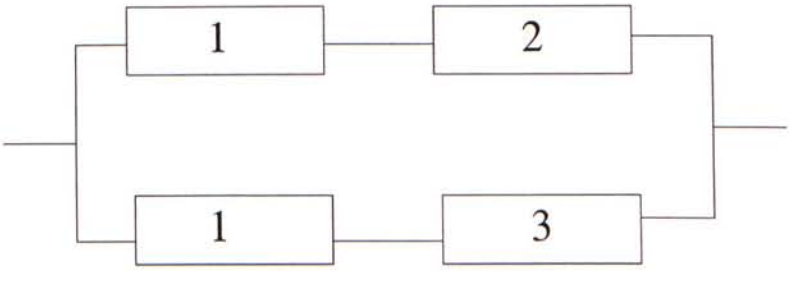


Figure 2: A network equivalent to Fig.1

2.3 Minimal cut sets

We have seen how using the minimal path sets to represent the systems as parallel arrangments of series networks in 2.2. We will now try to use the minimal cut sets to represent the systems as serial arrangements of parallel networks.

A state vector X is called a cut vector if $\phi(X) = 0$. If in addition to this, $\phi(Y) = 1$ for all $Y > X$, then X is said to be the minimal cut vector for the system. Since X and Y are binary vectors with their elements being either 0 or 1, therefore $Y > X$ implies that $y_i \geq x_i$, $i = 1, ..., m$, such that $y_i > x_i$ for some i. The set C is called the minimal cut set if it holds one of these minimal cut vector X's.

The minimal cut sets of a system holds the subsystems that is conceptually in series with one another. With all these subsystems, we can form another system that is equivalent to the original system by connecting them in series.

2.4 Structure functions

We will now define the structure functions for the systems. The structure functions are important to our studies, since the system reliabilities follow directly from the structure functions. Let's define a function $\phi(X)$ as stated

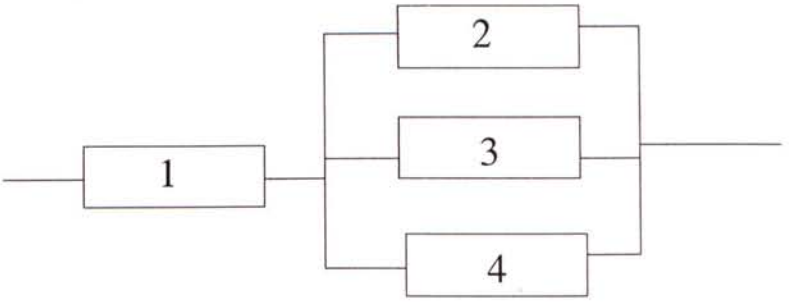


Figure 3: A network example to illustrate structure function

below:

$$\phi(X) = \begin{cases} 1, \textit{ if the system is functioning when the state vector is } X. \\ 0, \textit{ if the system fails when the state vector is } X. \end{cases}$$

$\phi(X)$ is known as the structure function of the systems. For a pure parallel network consists of n subsystems, $x_1, ..., x_n$, the structure function is given by $\phi(X) = max(x_1, ..., x_n) = 1 - \Pi_{i=1}^{n}(1 - x_i)$. While for a pure series structure consists of n subsystems, $x_1, ..., x_n$, the structure function is given by $\phi(X) = min(x_1, ..., x_n) = \Pi_{i=1}^{n} x_i$. We can often formulate the structure functions for the systems as the combinations of serial structures and parallel structures.

The structure functions can be formulated in three different ways: 1) to formulate it intuitively, 2) to formulate it with the help of minimal path sets, 3) to formulate it with the help of minimal cut sets.

We will first try to formulate the structure function intuitively. For the

structure function in Fig.3, we can see that there are two subsystems in series, one with component 1 as the only component, the other with components 2, 3 and 4 connecting in parallel. We can formulate it as $min(x_1, max(x_2, x_3, x_4))$, and the structure function becomes:

$$\begin{aligned}\phi(X) &= min(x_1, max(x_2, x_3, x_4)) \\ &= x_1 \times max(x_2, x_3, x_4) \\ &= x_1 \times (1 - (1 - x_2)(1 - x_3)(1 - x_4)) \\ &= x_1x_2 + x_1x_3 + x_1x_4 - x_1x_2x_3 - x_1x_2x_4 - x_1x_3x_4 + x_1x_2x_3x_4\end{aligned}$$

Now we will try to formulate the structure function with the help of minimal path sets. The exhaustive collection of minimal path sets for the system in Fig.3 are {1,2},{1,3},{1,4}, the structure function is:

$$\begin{aligned}\phi(X) &= max(x_1x_2, x_1x_3, x_1x_4) \\ &= 1 - (1 - x_1x_2)(1 - x_1x_3)(1 - x_1x_4) \\ &= x_1x_2 + x_1x_3 + x_1x_4 - x_1x_2x_3 - x_1x_2x_4 - x_1x_3x_4 + x_1x_2x_3x_4\end{aligned}$$

(consider the fact that x_i = either 0 or 1, $x_i^n = x_i^{n-1} = ... = x_i$)

Finally we will try to formulate the structure function with the help of minimal cut sets. The exhaustive collection of minimal cut sets for the system

are {1} and {2,3,4}. The structure function is:

$$\begin{aligned}\phi(X) &= min(max(x_1), max(x_2, X_3, x_4)) \\ &= max(x_1)max(x_2, x_3, x_4) \\ &= x_1 \times max(x_2, x_3, x_4) \\ &= x_1x_2 + x_1x_3 + x_1x_4 - x_1x_2x_3 - x_1x_2x_4 - x_1x_3x_4 + x_1x_2x_3x_4\end{aligned}$$

Please note that all the three methods give the same structure function for the same network.

2.5 The structure functions and the systems reliability

We have shown in the previous subsection how to derive the structure functions. The structure functions are of crucial importance in our research since we need using them to determine the systems reliablity functions for the different network configurations. The system reliability functions can be determined by the formula $R = P\{\phi(X) = 1\}$, where R is the overall system reliability and $\phi(X)$ is the structure function. We can take the network in Fig.3 as an illustrative example. The reliability function for the network in Fig.3 is $R = P\{\phi(X) = 1\}$, that is, $R = P\{(x_1x_2 + x_1x_3 + x_1x_4 - x_1x_2x_3 - x_1x_2x_4 - x_1x_3x_4 + x_1x_2x_3x_4) = 1\}$, which is equivalent to $R = r_1r_2+r_1r_3+r_1r_4-r_1r_2r_3-r_1r_2r_4-r_1r_3r_4-r_1r_2r_3r_4$. (Where r_i is the reliability of the i^{th} component, $i = 1, 2, 3, 4$). The reliability function of the network, R,

is now formulated as $R = r_1r_2+r_1r_3+r_1r_4-r_1r_2r_3-r_1r_2r_4-r_1r_3r_4-r_1r_2r_3r_4$. This example shows the importance of structure functions in formulating system reliabilities.

Since the structure function, $\phi(X)$, is a Bernoulli random variable, we may also compute the system reliability function R as the expected value of the structure function. Take the network in Fig.3 as the illustrative example again. The reliability function for can be expressed as $R = E[\phi(X)]$, this is equivalent to $R = E[x_1x_2+x_1x_3+x_1x_4-x_1x_2x_3-x_1x_2x_4-x_1x_3x_4+x_1x_2x_3x_4]$. Which is the same as $R = r_1r_2+r_1r_3+r_1r_4-r_1r_2r_3-r_1r_2r_4-r_1r_3r_4-r_1r_2r_3r_4$. Therefore the reliability function for the network is $R = r_1r_2 + r_1r_3 + r_1r_4 - r_1r_2r_3 - r_1r_2r_4 - r_1r_3r_4 - r_1r_2r_3r_4$.

With the structure functions, we can formulate the system reliability functions. With the system reliability functions we can investigate into the systems reliabilities.

3 Literature review

3.1 Introduction

There are several types of system reliability optimization problems in the literature. [5]

1. Optimum reliability allocation for a series system. This type of problems concerns with choosing the best component reliabilites $R_i = r_i$ so as to maximize the system reliability $max\, R = \Pi_{i=1}^{n} R_i$, subject to m constraints $\sum_{i=1}^{n} g_{ij}(R_i) \leq b_j$, where $i = 1, 2, ..., m$, $g_{ij}(R_i)$ is the j^{th} resource at stage i, and b_j is the total available i^{th} resource[4].

2. Optimum redundancy allocation. This type of problems concerns with choosing the number of redundant parallel components in each of subsystems of a complex network so as to maximize the system reliability function $R(X) = \phi[R(x_1), ..., R(x_k)]$ which is subjected to m constraints $\sum_{i=1}^{m} g_{ij}(X) \leq b_j$, where $i = 1, 2, ..., m$, $g_{ij}(X)$ is the j^{th} resource at stage i, and b_j is the total i^{th} resource that is available.

3. Combined reliability and redundancy allocation. This type of problems is the combination of (1) and (2). They concern with maximizing the system reliability function R by choosing the best component reliabilities as well as the number of redundant parallel components.

4. Cost minimization subject to reliability constraints. This type of problems concerns with minimizing the total system cost while maintaining the system reliability not less than a prespecified value. Their mathematical model is $min\, C = \sum_{i=1}^{n} C_i(m_i)$ subject to $R \geq R_0$.

5. Reliability/cost optimization of general system. This type of problems concerns with maximizing the system reliability function R, while at the same time minimizing the system cost function C.

We concentrate on the 2^{nd} problem, the optimum redundancy allocation problem, in our research. In the literature there are 3 types of solution approaching schemes for this type of problem. They are, 1) approximation schemes, 2) heuristic search schemes, and 3) exact solution schemes. We shall have a brief review on all these three techniques.

3.2 Approximation schemes

Federowicz and mazumdar [2] and Mirsa and Sharma [6], proposed to apply geometric programming schemes to the constrained redundancy optimization problems. They used the famous fact that arithmetic mean is always larger than or equal to the geometric mean to solve the problems. The original constrained redundancy problem is first convert to a primal problem [4] and a dual problem is formed from this primal problem. The dual problem is first

solved and then the optimal solution of the primal problem is obtained by the relationship between it and the dual problem, then the optimal redundancy assignment can be found from this primal problem. In most cases the optimal redundancy assignments found are non-integers and need to be converted to the nearest integer.

3.3 Heurisitic search schemes

Sharma and Venkateswaran [7] first proposed a heuristic method by using the fact that system unreliability $Q = 1 - \Pi_{i=1}^{n}(1 - q_i^{x_i})$ can be approximated by $Q \simeq \sum_{i=1}^{n}(q_i^{x_i})$. They successively reduce the system unreliability Q by each time adding one redundant component to the component with the highest unreliability $q_i^{x_i}$ if the constraints are not violated. This algorithm is able to find the solutions that are close to the boundary of the feasible region, but does not guarantee the solution to be globally optimal.

Misra [8] then proposed another heuristic method by considering the same fact that $Q \simeq \sum_{i=1}^{n}(q_i^{x_i})$ to deal with multiple linear constrained problems. It solves the problem by first estimating an optimum redundancy allocation scheme to each of the constraints. The algorithm then compares the system unreliabilites of the previously mentioned allocation schemes whenever they are not the same. One redundancy will be added to the schemes with lower

reliabilities, the component of which one more redundancy is to be added will be determined by checking the "desirability factor F", which is the ratio of (system reliability improvement)/(relative cost). This actually means that the component to be assigned one more redundancy should bear the maximum increase in reliability for a certain increase in the cost. The algorithm terminates when the redundancy allocation schemes for all the constraints are the same.

Shi [9] also proposed a heuristic method by considering the minimal path sets of the problem. (Please refer to section 2.2 for the information on minimal path sets). This algorithm first identifies all the minimal path sets by the algorithm proposed in [10]. It will then calculate the ratio $(\Pi_{i\in Pt}R_i(x_i)/(\sum_{i\in Pt}\sum_{j=1}^{k}(g_i^j(x_i)/kC^j)))$, which is actually (the multiple of reliabilities in the minimal path set)/(the percentage of consumed resources by this minimal path set). It will then choose the minimal path set with the largest ratio. All the components with the minimal path set will be evaluated according to the "selection-factor" ratio $b_i(x_i) = \frac{R_i(x_i)}{(\sum_{j=1}^{k}(g_i^j(x_i)/kC^j))}$. The component that bears the highest "selection factor" will be assigned one more redundancy in parallel. It will then remove the current minimal path set from further consideration whenever any constraint(s) is/are violated. The algorithm will then calculate for the most desirable minimal path set again and continue with the process until at least one constraint is exactly satisfied

and no other constraints are violated.

3.4 Exact solution schemes

Dynamic programming method is also used for solving constrained redundancy optimization problems. In most cases dynamic programming can only be used to solve for series-parallel network problems. This is because of the fact that dynamic programming method requires a separable problem structure. There are also limits on the number of constraints of the problem. The computation effort required to solve the problem will increase exponentially with the number of constraints if we use dynamic programming method to solve for it. Kettelle [11] proposed solving the constrained redundancy problem by using dynamic programming and the concept of dominating sequences. Tillman, Hwang and Kuo [4] proposed using lagrange multipliers to deal with the multiple constraints problems.

Branch and bound method was also developed. It finds the optimal solution by repeating partitioning the feasible solution space into mutually exclusive and exhaustive subsets and then establishes an upper bound over the objective functions within these subsets until the best solution is found.

Mirsa and Sharma [12] developed an efficient enumeration technique that searches for the solution of constrained redundancy problems in the neigh-

bour of the feasible region. This reduces the computational effort to find the solutions while comparing with other enumeration techniques. Unlike dynamic programming method, this techniques is applicable to all kinds of reliability network.

3.5 Software reliability

Berman and Ashrafi [13] presented four software reliability problem formulations to illustrate how to optimize the software reliability. These four problems are essentially combined reliability and redundancy allocation problems. One thing to notice is that as different from the hardware failures, software failures is defined as discrepancy between expected and actual output [14].

The author in [13] assumed that softwares performing the same function but with different version number will have different failure rates. These same-function softwares can therefore be used as redundancies to the system in the same way as hardware elements.

Sarper [15] later showed the possibility of solving the four software reliability models without developing special techniques as proposed in [13] by an optimization software called LINGO [16].

4 Characteristics of Series-parallel network

4.1 Series-parallel network problem formulation

The constrained redundancy optimization problem for series-parallel networks can be formulated as the following mathematical programming problem[17]:

$$max\ R(X) = \pi_{i=1}^{n}[1 - (1 - r_i)^{x_i}] \tag{1}$$

$$s.t.\ g_i(X) \leq b_i, i = 1, ..., m,\ X \in \mathbb{Z}^n, \tag{2}$$

where $R(X)$ is the overall system reliability, $X = (x_1, x_2, ..., x_n)$ is a redundancy assignment of the whole system, $r_i \in (0, 1)$ is the reliability of the component in the i^{th} subsystem in the system, x_i is the number of redundancies assigned to the i^{th} subsystem, $g_i(X), i = 1, ..., m$, are the resources constraints for the system, they can be linear or non-linear, $\mathbb{Z}^n$ is the positive integer vector set in $\mathbb{R}^n$.

We denote by the symbol S the feasible set of redundancy assignments of the problem. Without loss of generality, we assume that $r_1 \leq r_2 \leq ... \leq r_n$, thorough our discussion on the series-parallel networks, since this will make the discussion easier to be understood, but does not affect the system reliability assignment. We have, in fact, rearranged all the r_i's for the examples included in our discussion, if they are not in this order.

4.2 Characteristics of series-parallel networks

As investigated by Misra and Sharma [12] in 1991, the optimal redundancy assignments of all the reliability networks locate close to the boundary of the feasible region. This is because of the monotone property of the problems. This result is valid also for the series-parallel networks we investigated, since $R(X)$ and $g_i(X)$'s in (1),(2) also bear the same monotone property.

It had been shown that further to the monotonicity property, series-parallel networks optimal redundancy assignments also bear another important property, the maximal monotone property[17]. Sun and Li had proved that "*An Optimal redundancy assignment of* $(1)-(2)$ *must be both non-inferior and Maximal Monotone* " [17], which means that the optimal redundancy assignment: 1. must be close to the feasible boundary. 2. must satisfy the condition such that more redundancies are assigned to the less reliable components, if this is possible.

We investigate into the 2^{nd} optimal property mentioned above and are able to reveal some further properties.

4.3 Some further properties of Maximal Monotonicity

4.3.1 Definitions and the background

Definition 1 A feasible redundancy assignment X is said to be noninferior if there exists no other feasible redundancy assignment $y \in S$ such that $y_i \geq x_i, (i = 1, ..., n)$, with at least one strictly inequality[17].

Noninferior redundancy assignments are those redundancy assignments such that not a single extra redundancy component can be added to them if they are still to be feasible. They are the maximal feasible assignments. Because of the monotonicity property of the problem, the optimal redundancy assignment of any reliability network must be a noninferior assignment.

Definition 2 Let $X = (x_1, \ldots, x_{i-1}, x_i, \ldots, x_j, x_{j+1}, \ldots, x_n) \in S$, where x_i's are arranged in ascending order of reliability. S is the feasible set of redundancy assignment. We denote by $p_x(i, j) = (x_1, ..., x_{i-1}, x_i + 1, ..., x_j - 1, x_{j+1}, ..., x_n)$ the unit monotone transformation of x, if $i < j$ and $x_j \geq x_i + 1$.

The word "monotone" implicitly implies a redundancy assignment setting of $x_i \geq x_j$ if $j > i$. Unit monotone transformation always improves the overall system reliability, Sun and Li has proved this in [17].

Definition 3 A noninferior point X is said to be a maximal monotone point(MMP) if, for all $1 < i < j < n$, whenever $x_i < x_j$ implies that the unit monotone transformation of this point is not feasible.

In addition to being noninferior, it is clear that the optimal redundancy assignment must also be maximal monotone. If the candidate point is not maximal monotone, then an unit monotone transformation to it will be possible. Unit monotone transformations always improve the system reliabilities. Therefore the possibility of carrying out unit monotone transformations implies that the system reliability can still be further improved, this very redundancy assignment must not give the the highest system reliability, and it must not be the optimal redundancy assignment.

The maximal monotone property can be represented mathematically as "the optimal redundancy assignment in problem (1) – (2), X^*, must satisfy the condition such that, $\forall\, 1 \leq i < j \leq n : x_i^* < x_j^* \Longrightarrow p_{x^*}(i,j) \notin S$" [17].

We may use the maximal monotone property as a tool to eliminate the noninferior redundancy assignments that are not maximal monotone. Sun and Li[17] has developed an efficient search algorithm that identifies the maximal monotone noninferior points (the redundancy assignments can also be called *points*) from the noninferior sets and searches for the optimal solutions

within these maximal monotone noninferior sets.

Our further investigation shows that the maximal monotone points can still be classified into two different categories. We can confine our search space even more by checking only one of these two types of maximal monotone points for the optimal redundancy assignment. We will talk about this in greater detail in the following section.

4.3.2 The proper and the improper MMPs

Sometimes we can make a feasible redundancy assignment by assigning an extra redundancy assignment to the less reliable component while at the same time take away one redundancy from the more reliable component. Let's denote this by the term "*unit transformation*".

Definition 4 If the original point $X = (x_1, ..., x_{i-1}, x_i, ..., x_j, x_{j+1}, ..., x_n)$ then the Unit Transformation of X, is $P_{ut}(i, j) = (x_1, ..., x_{i-1}, x_i + 1, ..., x_j - 1, x_{j+1}, ..., x_n)$.

Unlike the "Unit monotone transformation", we do not require the condition such that $x_j \geq x_i + 1$ when i is smaller than j, if we are to carry out the "Unit transformation".

Though it can be argued that assigning more redundancies to the lower

reliability components may increases the overall system reliability(since it is intuitive that the 'bottleneck' of system reliability often lies on the less reliable components), but this is not necessary, and it can be shown that the unit transformation, as different from the unit monotone transformation, does not always improve the overall system reliability. We discovered that some of the maximal monotone points are assigning too many redundancies to the less reliable component(s). This is equivalent to carrying out too many unit transformations. Instead of increasing the overall system reliability, this decreases the overall system reliability.

We feel the need to identify from the other MMPs the maximal monotone points (MMPs) that have been assigning too many extra redundancies to the less reliable components. We denote this type of MMPs by the phrase "improper MMPs". The term "improper" implies this type of MMPs will not contribute to finding the optimal solutions in our redundancy assignment problems, as the system reliabilities achieved by these redundant assignments are lowered. In an improper assignment the more reliable component replaces the less reliable component to become the new system reliability 'bottleneck'.

If we take a closer look at the system reliability formulae, we can see that if the unit transformed MMP is to improve the overall system reliability, the

following inequality must hold.

$$(1 - b^{p+1})(1 - a^{q-1}) \geq (1 - b^{p})(1 - a^{q}) \tag{3}$$

where $a = (1 - r_a), b = (1 - r_b)$,

r_a = reliability index of a more reliable component,

r_b = reliability index of a less reliable component,

p is the original number of redundancies assigned to the less reliable component,

q is the original number of redundancies assigned to the more reliable component.

Definition 5 Improper MMPs are those MMPs that do not satisfy inequality (3). Proper MMPs are those MMPs that satsify inequality 3.

Inequality (3) refers to the situtations such that the system reliability achieved by assigning one more redundant path to the less reliable component and at the same time deducting one redundant path from the more reliable component is higher than the system reliability achieved by the original redundancy assignment (i.e $(1 - b^{p+1})(1 - a^{q-1}) \geq (1 - b^{p})(1 - a^{q})$).

This inequality will hold if an unit transformation to the original redundancy assignment improves the original system reliability.

But this inequality does not always hold. There are really some cases that it does not hold. A unit transformation reduces the overall system reliabilities in these cases, and the resulting MMPs will then become improper MMPs, as the overall system reliabilities are lowered. (The more reliable components now replace the less reliable components to become the new system 'bottlenecks'.)

Usually, when p is too large and q is too small(i.e. the MMP has been assigning too many redundancy to the less reliable components), the above inequality will no longer hold. If any MMP has undergone the unit transformation without satisfying inequality (3), the resulting MMP (if it is a MMP) will be an improper MMP.

We can improve the overall system reliability in such cases by reversing the process of unit transformation on these improper MMPs (i.e. assigning one more redundancy to the more reliable component and at the same time decreasing the number of redundancies for the less reliable component by one).

We will investigate the conditions such that reversing the unit transformations improve the overall system reliabilities. This is equivalent to identifying the conditions for the MMPs to be improper. This is also equivalent to identifying when the more reliable component will replace the less reliable component to become the new 'bottleneck'.

We can see that if we are to conduct a reverse transformation in order to improve the overall reliability, inequality (4) must hold. Any improper MMP will therefore satisfy inequality (4).

$$(1-b^{p-1})(1-a^{q+1}) \geq (1-b^{p})(1-a^{q}) \tag{4}$$

Where $a < b$ and $q < p$. a, b, are rational numbers while q, p are positive integers. let's assume $p = q + n$, and substitute this to (4), we have:

$(a^q + b^p - a^{q+1} - b^{p-1} + b^{p-1}a^{q+1} - b^p a^q) \geq 0$

substitute $\frac{a}{b} = c$ into the above inequality we then have:

$$(b^q c^q + b^{q+n} - b^{q+1}c^{q+1} - b^{q+n-1} + b^{q+n-1}b^{q+1}c^{q+1} - b^{q+n}b^q c^q) \geq 0$$

$$b^{n-1}[(b-1) + (c-1)b^{q+1}c^q] + (1-bc)c^q \geq 0$$

$$b^{n-1} \leq -\frac{(1-bc)c^q}{(b-1)+(c-1)b^{q+1}c^q}$$

$$(n-1)\log(b) \leq \log[\frac{(1-bc)c^q}{(1-b)+(1-c)b^{q+1}c^q}]$$

$$n-1 \geq \frac{\log[(1-bc)c^q]}{\log(b)} - \frac{\log[(1-b)+(1-c)b^{q+1}c^q]}{\log(b)}$$

$$\Rightarrow n \geq \frac{\log[(1-a)(\frac{a}{b})^q] - \log[(1-b)+(1-\frac{a}{b})b^{q+1}(\frac{a}{b})^q]}{\log(b)} + 1 \tag{5}$$

$$\Rightarrow n \geq (\frac{\log(\frac{(1-a)}{((1-b)+(b-a)a^q)}) + q\log(\frac{a}{b})}{\log(b)}) + 1 \tag{6}$$

Therefore in a redundancy assignment, if p is larger than q for n, conducting a reverse unit transformation will increase the overall system reliability. Physically the above inequality means that if too many redundancies are

assigned to the less reliable component, then the overall system reliabilities will be lower because the more reliable component now becomes the new 'bottleneck'. The n we found here is the maximum value for the difference between the redundancy levels before the more reliable component becomes the new system 'bottleneck'.

Definition 6 If any MMP that has a difference between redundancy levels greater than the right-hand side expression in (6), then this MMP is an improper MMP.

The lower bound of n can be found by (6). We shall show that the function on the right hand side of the "$\leq$" symbol of the inequality (6) actually increases with b and decreases with a, where $b = (1 - r_b)$, $a = (1 - r_a)$, $r_a > r_b$, $b > a$. Consider a function $f(a, b, q)$ defined by:

$$f(a,b,q) = \frac{\log(\frac{(1-a)}{((1-b)+(b-a)a^q)}) + q\log(\frac{a}{b})}{\log(b)} + 1$$

We have the following theorem:

Theorem 1: $f(a, b, q)$ is a decreasing function of a.

Proof:

$$\frac{\partial f}{\partial a} = (\frac{1}{\log(b)})[\frac{(q - qa - a)}{(a(1-a))} - \frac{a^{q-1}(bq - qa - a)}{(1 - b + ba^q - a^{q+1})}]$$

We first let $f_1 = q - qa - a + a^{q+1}$. We need to prove that f_1 is larger than zero, since we need to use this condition in our later proof.

$$\begin{aligned} f_1 &= q - qa - a + a^{q+1} \\ &= q(1-a) - a(1-a^q) \\ &= [q - \frac{a(1-a^q)}{(1-a)}](1-a) \\ &= [(1-a) + \ldots + (1-a^q)](1-a) \\ &> 0 \, (\because \; 0 < a < 1) \\ &\Rightarrow \; f_1 > 0 \end{aligned} \tag{7}$$

Let $f_2 = [\frac{(q-qa-a)}{(a(1-a))} - \frac{a^{q-1}(bq-qa-a)}{(1-b+ba^q-a^{q+1})}]$

Let's prove that f_2 is positive. Since $\frac{\partial f(a,b,q)}{\partial a} = f_2 \times (\frac{1}{\log(b)})$, therefore if $f_2 > 0$, we can then prove $f(a,b,q)$ is a decreasing function of a. To prove $f_2 > 0$ is actually equivalent to proving $f_3 = (q-qa-a)(1-b+ba^q-a^{q+1}) + (-a^q + a^{q+1})(qb - qa - a) > 0$ (where $1 > b > a > 0$).

So let's first prove $f_3 > 0$:

$$\begin{aligned}
f_3 &= (q - qa - a)(1 - b + ba^q - a^{q+1}) + (-a^q \\
&\quad +a^{q+1})(qb - qa - a) \\
&= q - qb - qa + qba - a + ab - ba^{q+1} + a^{q+1} \\
&= q(1-b)(1-a) - a(1 - b - a^q + ba^q) \\
&= (1-b)(q - qa - a + a^{q+1}) \\
&= (1-b) \times f_1 \\
&> 0 \\
&\quad (because\ of\ (7)\ and\ also \\
&\quad because\ (1-b) > 0) \\
\Rightarrow\ & f_2 > 0 \\
&\quad (since\ this\ is\ equivalent\ to\ f_3 > 0,)
\end{aligned}$$

Because $\frac{\partial f(a,b,q)}{\partial a} = f_2 \times (\frac{1}{\log(b)})$, therefore we have $\frac{\partial f(a,b,q)}{\partial a} < 0$ (since $(\frac{1}{\log(b)}) < 0$ and $f_2 > 0$). In other words, $f(a, b, q)$ is a decreasing function of a, since the partial differentiation is less than zero and this completes our proof for Theorem 1 . Theorem 1 means that the lower bound n in inequality (6) decreases with a.

Theorem 2: $f(a,b,q)$ is an increasing function of b.

Proof: Let $f^*(a,b,q) = \log(\frac{a^q - a^{q+1}}{b^q}) - \log(1-b+ba^q-a^{q+1})$ and let $b_2 > b_1$.

We now have: $f^*(b_2) - f^*(b_1) = \log\{(\frac{b_1}{b_2})^q \frac{[(1-b_1)+(b_1-a)a^q]}{[(1-b_2)+(b_2-a)a^q]}\}$

$$
\begin{aligned}
\textit{since} \quad & {b_1}^q[(1-b_1)+(b_1-a)a^q] - {b_2}^q[(1-b_2)+(b_2-a)a^q] \\
= \; & -{b_1}^q(b_1-1) - {b_2}^q(1-b_2) - {b_1}^q a^q(a-b_1) - {b_2}^q a^q(b_2-a) \\
< \; & -{b_1}^q(b_1-1) - {b_1}^q(1-b_2) - {b_1}^q a^q(a-b_1) - {b_1}^q a^q(b_2-a) \\
= \; & -{b_1}^q(b_1-b_2) - {b_1}^q a^q(b_2-b_1) \\
< \; & -{b_1}^q(b_1-b_2) - {b_1}^q(b_2-b_1) = 0 \\
\Rightarrow \; & {b_1}^q[(1-b_1)+(b_1-a)a^q] - {b_2}^q[(1-b_2)+(b_2-a)a^q] < 0 \\
\Rightarrow \; & \frac{{b_1}^q[(1-b_1)+(b_1-a)a^q]}{{b_2}^q[(1-b_2)+(b_2-a)a^q]} < 1 \\
\Rightarrow \; & \log\{(\frac{b_1}{b_2})^q \frac{[(1-b_1)+(b_1-a)a^q]}{[(1-b_2)+(b_2-a)a^q]}\} < 0 \\
\Rightarrow \; & f^*(b_2) - f^*(b_1) < 0 \qquad (*)
\end{aligned}
$$

The original function $f(a,b,q)$ is equal to $\frac{f^*(a,b,q)}{\log(b)} + 1$, and from $(*)$, we have

: $f(a,b_2,q) - f(a,b_1,q) = \frac{f^*(b_2) - f^*(b_1)}{\log(b)}$

$= \frac{-ve}{-ve}$

> 0

therefore $f(b)$ is an increasing function of b.

From theorems 1 and 2, we know that the extreme value of the lower bound n occurs at the the place with the smallest a and the largest b.

We now need to show that the expression $\frac{\log[\frac{(1-a)}{(1-b)+(b-a)a^q}]}{\log(b)}$ is always a negative Number. We will use this result to further simplify the polynomial expression in (6) so as to derive a simpler expression for n.

Theorem 3: $\frac{\log[\frac{(1-a)}{(1-b)+(b-a)a^q}]}{\log(b)}$ is always less than zero.

Proof:

$$
\begin{aligned}
& (1-a)-(1-b)-(b-a)a^q \\
= \; & (b-a)-(b-a)a^q \\
= \; & (b-a)(1-a^q) \\
> \; & 0 \\
\Rightarrow \; & (1-a) > (1-b)+(b-a)a^q \\
\Rightarrow \; & \frac{(1-a)}{(1-b)+(b-a)a^q} > 1 \\
\Rightarrow \; & \log[\frac{(1-a)}{(1-b)+(b-a)a^q}] > 0 \\
\Rightarrow \; & \frac{\log[\frac{(1-a)}{(1-b)+(b-a)a^q}]}{\log(b)} < 0 \\
& (since \; \log(b) < 0)
\end{aligned}
$$

For easier computation, the polynomial expression in (6) can now be further simplified to (8) with not much loss of precision:

$$\frac{q\log(\frac{a}{b})}{\log(b)} + 1 \tag{8}$$

Inequality (6) now becomes:

$$n \geq \frac{q \log(\frac{a}{b})}{\log(b)} + 1 \tag{9}$$

One thing we should notice is that (9) is always an over-estimation of the real bound n, this is because of the fact that $\frac{\log[\frac{(1-a)}{(1-b)+(b-a)a^q}]}{\log(b)}$ is always less than zero. Though (9) might not be as effective as the original inequality, we can always use it, and the new inequality is obviously more efficient than the original one to identify the improper MMPs.

With this simpler inequality (9) we can formulate an algorithm to test for the improper maximal monotone points.

Before we develop the algorithm that utilizes our findings, we need to decide the policy to test for the different redundancy assignment pairs.

Suppose we have a system composed of three components whose reliabilities are r_b, r_{a_1} and r_{a_2}, repectively (where $r_b < r_{a1} < r_{a2}$), and suppose if we need to test for a system configuration of (p, q_1, q_2), whereas p, q_1, q_2 are the number of redundancies assigned to the components with reliabilities r_b, r_{a1} and r_{a2} repectively. We should select q_2 as the number to test for if the following inequality holds

$$p - q_2 - q_2 \times (\frac{\log(\frac{a_2}{b})}{\log(b)}) \quad > \quad p - q_1 - q_1 \times (\frac{\log(\frac{a_1}{b})}{\log(b)})$$

This is because a larger difference between the level of redundancies and the lower bound we found in (9) is more favourable to our testing procedure, since this will increase the chance for a successful point elimination.

Let's now continue with the previous inequality to find a general guideline from it. After some rearrangement of terms the above inequality now becomes:

$$\begin{aligned}
p - q_2 - p + q_1 &> q_2 \times (\frac{\log(\frac{a_2}{b})}{\log(b)}) - q_1 \times (\frac{\log(\frac{a_1}{b})}{\log(b)}) \\
q_1 - q_2 &> q_2 \times (\frac{\log(a_2) - \log(b)}{\log(b)}) - q_1 \times (\frac{\log(a_1) - \log(b)}{\log(b)}) \\
0 &> q_2 \times (\frac{\log(a_2)}{\log(b)}) - q_1 \times (\frac{\log(a_1)}{\log(b)}) \\
q_2 \times \log(a_2) &> q_1 \times \log(a_1)
\end{aligned}$$

$$\Rightarrow \frac{\log(a_1)}{\log(a_2)} > \frac{q_2}{q_1} \tag{10}$$

We should select q_2 to be tested if inequality (10) holds. We should otherwise select q_1 if the above inequality does not hold.

With all these results, we can now derive a search algorithm that uses our new knowledge to confine the search space for searching the optimal redundancy assignment for series-parallel systems. This search algorithm is modified from the MIP algorithm proposed by Misra and Sharma in 1991[12] and also the Maximal-Monotonicity-checking algorithm proposed by Sun and Li and in 1998[17]. But we further improve the efficiency by eliminating the

improper Maximal Monotone points (improper MMPs). It separates the normal MMPs from the improper MMPs and discards the improper MMPs from reliability evaluations.

This algorithm is modified from the algorithms in [17] and [12]:

The Algorithm

0) Calculate for all $\log(1-r_1), \log(1-r_2), \ldots \log(1-r_n)$. Determine the upper bound ${x_i}^u$ and lower bound ${x_i}^l$ of x_i for $i = 1, 2, ..., n$. Set $x = ({x_1}^u, {x_2}^l, ..., {x_n}^l), X^* = X, t = 2, l = 0$, goto VI.

I) If X is a proper MMP then goto VII.

II) $x_2 = x_2 + 1$.

III) If $(1, x_2, ..., x_n) \in S$ then find ${x_1}^{max}$ such that $({x_1}^{max}, x_2, ..., x_n) \in S$, set $X = ({x_1}^{max}, x_2, ..., x_n)$, goto VI.

IV) Set $l = l + 1$, if $l > n - 2$ goto VIII.

V) Set $k = t + l, x_k = x_k + 1$, if $x_k > {x_k}^u$ then goto IV. Otherwise $x_j = {x_j}^l$ for $j = 2, ..., k-1, l = 0$ goto step III.

VI)

1) If the point is a MMP and $p - q > 2$ for this point, use inequality (5) to test for the n. If there are more than one pair of p,q to be compared, we need to consider the following three cases:

a) If there are two high reliability components competing and both of them have q redundancy assignments, choose the one with lower reliability, since functions in (4), (5) are both decreasing function of r_a, therefore a lower r_a will yield us a smaller n, which gives us a smaller bound.

b) If there are two high reliability components competing and the one with the highest reliability has more redundancy assignment than the one with the second highest reliability, then we should always choose the less reliable component(instead of the more reliable one, the reason is same as in (a)).

c) If there are two high reliability components competing, and the more reliable component has less components than the less reliable component. We should then choose the more reliable component to test for, if $\frac{\log(1-r_{a1})}{\log(1-r_{a2})} \geq \frac{q_2}{q_1}$ where r_{a1}, r_{a2} are the reliabilities for the second most reliable and the most reliable components, while q_1, q_2 are the number of redundancy assignment for the second most reliable and the most reliable components respectively.

2) If $n > p - q$ then keep this point ; If $n < p - q$ and the unit reverse transformed point is a feasible point, then we can drop out the original maximal monotone point from further consideration. Goto II.

VII) Compute $R(X)$ and set $x^* = x$ if $R(X) > R(X^*)$. Goto II.

VIII) Stop.

4.4 Examples

We have implemented the alogrithm described in the previous section into a C programme that is enclosed in the appendix section(Chapter 8) to the end of this thesis.

Four different problems have been tested with our new algorithm, Let's first give these four example problems.

Example 1:[17] Consider a series-parallel system with 4 subsystems, $r = (0.65, 0.70, 0.75, 0.80)$, and two linear constraints, $g_1(X) = 6x_1 + 4x_2 + 3x_3 + 2x_4 \leq 30$ and $g_2(X) = 9x_1 + 4x_2 + 4x_3 + 3x_4 \leq 40$.

Example 2: [12] Consider the following constrained redundancy optimization problem.

$$
\begin{aligned}
&max\ \Pi_{i=1}^{5}(1-(1-r_i)^{x_i}) \\
&subject\ to\ c(X) = \sum_{i=1}^{5} c_i x_i \leq C, \\
&x \in \mathbb{Z}^5,
\end{aligned}
$$

where $r = (0.7, 0.75, 0.8, 0.85, 0.9)$, $c = (2, 2, 3, 3, 1)$ and $C = 20$.

Example 3: [18] Consider the following constrained redundancy opti-

mization problem.

$$
\begin{aligned}
& max\, \Pi_{i=1}^{5}(1-(1-r_i)^{x_i}) \\
& subject\, to\, c(X) = \sum_{i=1}^{5} c_i x_i \leq C, \\
& w(X) = \sum_{i=1}^{5} w_i x_i \leq W, \\
& x \in \mathbb{Z}^5,
\end{aligned}
$$

where $r = (0.65, 0.75, 0.80, 0.85, 0.9)$, $c = (9, 4, 7, 7, 5)$, $w = (6, 9, 7, 8, 8)$, $C = 100$, $W = 104$.

Example 4: [12] Consider the following constrained redundancy optimization problem.

$$
\begin{aligned}
& max\, \Pi_{i=1}^{5}(1-(1-r_i)^{x_i}) \\
& subject\, to\, p(X) = \sum_{i=1}^{5} p_i x_i^2 \leq P, \\
& c(X) = \sum_{i=1}^{5} c_i (x_i + exp(x_i/4)) \leq C, \\
& w(X) = \sum_{i=1}^{5} w_i x_i exp(x_i/4) \leq W, \\
& x \in \mathbb{Z}^5
\end{aligned}
$$

where $r = (0.65, 0.75, 0.8, 0.85, 0.9)$, $p = (4, 2, 1, 2, 3)$, $c = (9, 4, 7, 7, 5)$, $w = (6, 9, 7, 8, 8)$. Five sets of (P,C,W) need to be considered: a) (110,175,200); b) (114,185,212); c) (116,190,218) ; d) (116,145,236); e(90,195,256).

4.5 Computational results

The computational results for the examples 1-4 are listed in the table below.

Example	$N_{feasible}$	$N_{non-inferior}$	$N_{maximal\ monotone}$	$N_{proper\ maximal\ monotone}$
1	42	22	9	3
2	157	48	11	7
3	1415	422	17	1
4a	494	99	5	2
4b	600	129	6	2
4c	642	120	7	2
4d	629	145	13	3
4e	665	106	12	3

$N_{feasible}$ = total number of feasible points,

$N_{non-inferior}$ = total number of non-inferior points,

$N_{maximal\ monotone}$ = total number of maximal monotone points,

$N_{proper\ maximal\ monotone}$ = total number of proper maximal monotone points.

The detailed computational results are enclosed in the appendix (Chapter 8).

4.6 New progress

We derived an equation(equation 8), which governs the upper bounds of the differences between the redundancy levels. We further simplify the equation as follows:

$$\frac{\log(\frac{a}{b})}{\log(b)} + 1 < n \tag{11}$$

$$\log(a) - \log(b) > (n-1)\log(b) \tag{12}$$

$$\log(a) > n\log(b) \tag{13}$$

$$a > b^n \tag{14}$$

From the above we know that inequaility $\frac{\log(\frac{a}{b})}{\log(b)} + 1 < n$ is actually equivalent to $a > b^n$. We can now replace inequality (9) by the inequality $a > b^n$ and use it to identify the improper MMPs. We can further simplify the checking process by building a "YES/NO" look-up table which considers the inequality $a > b^n$. Whenever the inequality holds then we put a "Y" in a corresponding place within the table(i.e. $a > b^n$ holds) , otherwise we put a "N" there(i.e. $a > b^n$ does not hold). The we just need checking the table to determine whether a MMP is improper.

let's now consider a simple example with reliabilities vector (r_a, r_{b_1}, r_{b_2}) to illustrate how building a table. We need first finding the gobal upper bound for the redundancy levels, this ensures that we will do the comparsions only

for those MMPs that need them. Any MMPs with larger differences will not require this checking, since they will automatically satisfy the inequality $a > b^n$. One thing to notice is that we need calculating the gobal upper bounds and building new tables for every of the different problems, since they may have different problem structures. The gobal upper bound can be calculated as $n_{upperbound} = \frac{\log(\frac{a_{min}}{b_{max}})}{\log(b_{max})} + 1$. Where $a = 1 - r_a$, $b = 1 - r_b$, a_{min} is the smallest value for a, and b_{max} is the largest value for b. We use a_{min} and b_{max} to determine the $n_{upperbound}$ because we know from our previous proofs that the $n_{upperbound}$ occurs when a is the smallest and b is the largest. if $n_{upperbound}$ is not an integer, then it should be rounded to the largest integer (i.e. $\ulcorner n_{upperbound} \urcorner$).

suppose $n_{upperbound}$ is equal to 3 in our illustrative example. We may build a table as follows:

	b^2	$b^{n_{upperbound}} = b^3$	c^2	$c^{n_{upperbound}} = c^3$
a	Y	Y	N	Y
b	NA	NA	N	Y

Where the y-axis corresponding to the more reliable components in our comparsion (i.e. a corresponding to the component with reliability r_a), x-axis corresponding to the less reliable components and the difference level

(i.e. b^2 refer to the component with the reliability r_b, and at the same time the redundancy difference between this component and the more reliable component is 2, i.e. n=2). If there is a "Y" at (b^2, a) that means $a > b^2$ (which means $n = 2 > \frac{\log(\frac{a}{b})}{\log(b)} + 1$).

To illustrate how using the table. Let's suppose that we have an maximal monotone redundancy assignment of $(5, 3, 4)$. According to the algorithm described in section 4.3, we pick the first two components to compare. The difference between the less reliable component assignment and the more reliable component assignment is 5-3 = 2. We can use the table to check if $a > b^n$ (i.e. $n > \frac{\log(\frac{a}{b})}{\log(b)} + 1$) holds under this assignment. We check the value at (b^2, a) in the table, we find a "Y" there, that means our redundancy assignment satisfies the inequality $a > b^n$ and we can reverse transform this redundancy assignment to see whether it is improper or not.

It can be seen that this look-up table simplifies the calculations involved in identifying the improper MMPs.

5 Extensions for the series-parallel reducible networks

In the last chapter, we have investigated some special properties of the series-parallel networks. But we have not discussed about the pure parallel networks yet. We will talk about them in the following.

Pure parallel networks are actually equivalent to the series-parallel networks in finding the maximal monotonic assignments. The redundancy assignments for this type of network is trivial from the results achieved in [17]. We can formulate the problem in the same way as what we did for the series-parallel network by changing all the reliabilities to unreliabilties. Instead of finding the best redundancy assignment by maximizing the overall system reliability, we need to minimize the overall system unreliability (this is equivalent to maximizing the overall reliability), in order to find the best redundancy assignment. Therefore, as opposite to what is true in the series-parallel networks, more redundancies should be assigned to the more reliable components in pure parallel networks .

Now we are going to extend part of the results of Sun and Li's work [17] and try to apply it onto the different series-parallel reducible networks. This is the major thing we are going to do in this chapter.

As we have introduced in the last chapter, Sun and Li [17] discovered that

in a pure series-parallel system, more redundant assignment should be made to the less reliable components. This result is utilized to confine the search space. We discovered some similar properties for the series-parallel reducible networks. We also propose using our new finding to confine the search space for optimal redundancy assignment of the series-parallel reducible networks.

5.1 Some new notations for computation

Before we go into the details of our new findings, we need to present some new notations that will appear in the later sections.

5.1.1 Notation

Since we are now discussing the redundancy assignments for the series-parallel reducible networks, it is important if we can use simple notations to describe the ways how the different components are connected with one another (i.e. in parallel or in series). This will enable the readers to better understand the physical structure of each of the series-parallel reducible networks. This is also important for developing our new algorithm, as the structure of the problem is vital for the algorithm to find the optimal solution.

We shall use the square brackets "[]" to denote two components or subsystems that are in parallel, and we shall use the round brackets "()" to

denote two components or susbsystems that are in series.

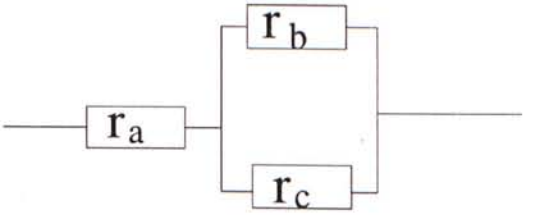


Figure 4: a simple series-parallel reducible network

For the simple series-parallel reducible network as shown in Fig.4, we represent by $X = (x_{r_a}, [x_{r_b}, x_{r_c}])$ the redundancy assignment of the network, where X is the redundancy assignment of the whole network, x_{r_a} is the number of redundancy assignment for the component with reliability r_a in Fig.4 , x_{r_b} and x_{r_c} are the redundancy assignments for components with reliability r_b and r_c, respectively.

Since the components with reliabilities r_b and r_c are connected in parallel, the redundancy assignments for them, x_{r_b}, x_{r_c}, are therefore bracketed by a square bracket. The remaining component (with reliability r_a) is in series with the aforesaid two components, it is therefore bracketed by a round bracket with the first two components.

This type of notations are capable of representing series-parallel network configurations easily and will enable us programming our new algorithm into

a computer programme.

5.2 Problem formulation

The constrained redundancy optimization problem for series-parallel reducible system is as follows:

$$max\, R(X) \tag{15}$$

$$s.t.\, g_i(X) \leq b_i, i = 1, ..., m,\ X \in \mathbb{Z}^n, \tag{16}$$

Where $R(X)$ is the overall system reliability under a specific redundancy assignment X, X is a redundancy assignment for the whole system, $g_i(X) \leq b_i,\ i = 1, ..., m$ are the resources constraints for the system, which can be linear or non-linear, $\mathbb{Z}^n$ is the positive integer vector in $\mathbb{R}^n$.

5.3 The series-parallel reducible networks

Since the objective function for system reliability, $R(X)$, is a monotonic function that increases with the reliability(ies) of its subsystem(s), therefore we investigate into the simplest kind of series-parallel network configurations and try to improve the reliability of these configurations so as to improve the overall system reliability. We still assume that all the redundancy assignments Xs' are close enough to the feasible frontier such that not a single extra redundancy can be assigned to any of the components in X. We can

ensure this by considering those redundancy assignments generated by the MIP algorithm [12], since they are, by definition the redundancy assignments that do not allow adding any extra redundancy assignments.

The simplest series-parallel reducible networks consists of three components is shown below in Fig.5. The reliabilities for the components A, B, and C are r_a, r_b, r_c, respectively. We assumed that $r_a > r_b > r_c$.

We shall now investigate how determining the maximal monotonic assignments in series-parallel reducible networks.

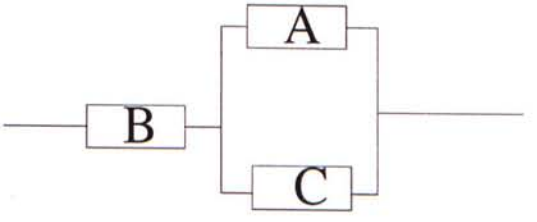


Figure 5: One of the simplest series-parallel reducible networks

We will consider the simplest kind of series-parallel reducible networks in Fig.5.

For the networks as shown above in Fig.5, we wish to know which components out of the three should keep the most redundancies. We may first divide the above network into two parts, the first part contains component B, whilst the second part contains components A and C connecting in parallel. For the second part of the network, this is a pure parallel network , we

know from our previous discussion that more redundancy should be put to the component with a higher reliability, so component A should have more redundancies than component C. Component A dominates the parallel part of the network. We then need to check if component A dominates the whole network. We take a close look of the system reliability equation, and discover that if component A is to dominant the whole network, the following inequality must hold:

$$(1 - b^p)(1 - a^q c^r) - (1 - b^q)(1 - a^p c^r) \leq 0$$

where $a = 1 - r_a$, $b = 1 - r_b$, $r_a > r_b$, $p = q + n$, and p, q, n, are all integers larger than zero.

If the above inequality does not hold, then this implies that instead of component A, component B dominates the whole network. In other words the largest number of redundancies should be put to component B, if the inequality does not hold.

Theorem 4: For the type of network as shown above in Fig.5, more redundancies should be put to the component B.

Proof :

$$
\begin{aligned}
& (1-a^p)(1-a^q c^r) - (1-a^q)(1-a^p c^r) \\
= \; & -a^p - a^q c^r + a^p c^r + a^q \\
= \; & a^q(1-a^n)(1-c^r) \\
\geq \; & 0 \qquad (since\ (1-a^n)\ and\ (1-c^r)\ both\ \geq\ 0) \\
\therefore \; & (1-a^p)(1-a^q c^r) - (1-a^q)(1-a^p c^r) \geq 0 \\
\therefore \; & (1-a^p)(1-a^q c^r) \geq (1-a^q)(1-a^p c^r) \\
\therefore \; & \frac{(1-a^p)}{(1-a^q)} \geq \frac{(1-a^p c^r)}{(1-a^q c^r)}
\end{aligned}
$$

$$
\begin{aligned}
& since\ \frac{(1-b^p)}{(1-b^q)} \geq \frac{(1-a^p)}{(1-a^q)} \quad (from\ Lemma\ 1\ of\ [17]) \\
\Rightarrow \; & \frac{(1-b^p)}{(1-b^q)} \geq \frac{(1-a^p c^r)}{(1-a^q c^r)} \\
\Rightarrow \; & (1-b^p)(1-a^q c^r) \geq (1-b^q)(1-a^p c^r) \\
\Rightarrow \; & (1-b^p)(1-a^q c^r) - (1-b^q)(1-a^p c^r) \geq 0
\end{aligned}
$$

where r_a, r_b, r_c are the reliabilities for components A, B and C respectively, $r_a > r_b > r_c$, and $a = 1 - r_a$, $b = 1 - r_b$, $c = 1 - r_c$, $p = q + n$.

Therefore putting more redundancies to component B yields a higher overall system reliability and this completes our proof for Theorem 4.

For the 3-component network as shown in Fig.6, we can also first divide

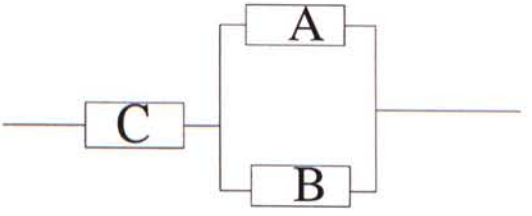


Figure 6: One of the simplest series-parallel reducible networks

it into two parts. The first part contains component C, while the second part contains components A and B connecting in parallel. Again we know that for the second part of the network, we need putting more redundancies to component A since it has a higher reliability.

Now we need to check whether component A or component C deserves more redundancies. Again we take a close look at the system reliability function, we discovered that if component A deserves more redundancies then the following inequality must hold:

$(1 - c^q)(1 - a^p b^r) - (1 - c^p)(1 - a^q b^r) \leq 0$

where $a = 1 - r_a$, $b = 1 - r_b$, $r_a > r_b$, $p = q + n$, and p, q, n, are all integers larger than zero.

Theorem 5: For the type of network as shown in Fig.6, more redundancies should be put to the component C.

proof:

$$
\begin{aligned}
& (1-a^p)(1-a^q c^r) - (1-a^p)(1-a^q c^r) \\
= \;& a^q(1-a^n)(1-c^r) \\
\geq \;& 0 \\
\therefore \;& (1-a^p)(1-a^q c^r) - (1-a^p)(1-a^q c^r) \geq 0 \\
\therefore \;& \frac{(1-a^p)}{(1-a^q)} \geq \frac{(1-a^p c^r)}{(1-a^q c^r)}
\end{aligned}
$$

$$
\begin{aligned}
& since \; \frac{(1-c^p)}{(1-c^q)} \geq \frac{(1-a^p)}{(1-a^q)} \; (from \; Lemma \; 1 \; of \; [17]\,) \\
\Rightarrow \;& \frac{(1-c^p)}{(1-c^q)} \geq \frac{(1-a^p c^r)}{(1-a^q c^r)} \\
\Rightarrow \;& (1-c^p)(1-a^p c^r) \geq (1-c^q)(1-a^p c^r) \\
\Rightarrow \;& (1-c^p)(1-a^q c^r) - (1-c^q)(1-a^p c^r) \geq 0
\end{aligned}
$$

where r_a, r_b, r_c are the reliabilities for components A, B and C respectively, $r_a > r_b > r_c$, and $a = 1 - r_a$, $b = 1 - r_b$, $c = 1 - r_c$, $p = q + n$.

Therefore component C deserves more redundancies, and this completes our proof for theorem 5.

Now let's consider the last variation of the 3-component network as shown in Fig.7. We need to consider which of the components A and B should have more redundancies. We can see that if the following inequality holds, then

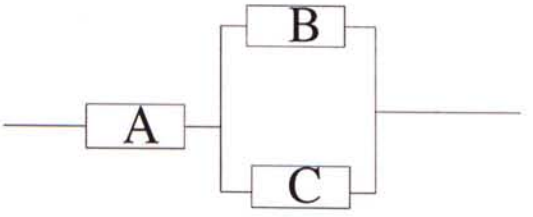


Figure 7: One of the simplest series-parallel reducible networks

we should put more redundancies to component B, otherwise we should put more redundancies to component A: $(1-a^q)(1-b^p c^r)-(1-a^p)(1-b^q c^r) \geq 0$

Theorem 6: For the type of network as shown in Fig.7, it is unclear which component deserves more redundancies.

Proof:

Let $p = 3,\ q = 2,\ r = 1,\ n = 1$

case 1: If $a = 0.79,\ , b = 0.8,\ c = 0.81$ then substituting these values into the previous inequality will give us: $(1 - 0.79^2)(1 - 0.8^3 \times 0.81) - (1 - 0.79^3)(1 - 0.8^2 \times 0.81) = -0.0241 \leq 0$

case 2: If $a = 0.4,\ b = 0.6,\ c = 0.9$ substituting these values into the previous inequality will however give us: $(1 - 0.4^2)(1 - 0.6^3 \times 0.9) - (1 - 0.4^3)(1 - 0.6^2 \times 0.9) = 0.0440 \geq 0$

So it is unclear whether component B or component A deserves more redundancies.

But it is clear that we can derive an additional theorem from the theorems 4-6.

Theorem 7: For the 3-component networks as shown in Fig.5 - Fig.7, we should put more redundancies to the single component that is in series with the other two components if this single component does not bear the highest reliability.

proof:

The result follows directly from theorems 4-6.

Since the system reliability function R(X) is a monotonic function that increases with the reliability(ies) of the subsystem(s), increasing the reliability of a subsystem will also increase the reliability of the whole system. As a result we are able to develop an efficient algorithm that search for the optimal redundancy assignment for the series-parallel reducible networks from Theorem 7. This algorithm works for series-parallel networks which bear the 3-component network structures but do not bear any additional networks that are parallel to the 3-component networks.

5.4 The algorithm

[17][12]

0) Calculates an upper bound x_i^u and lower bound x_i^l for all the decision variables. The lower bounds are normally $x_i^l = 1$ unless otherwise stated, and the upper bounds x_i^u can be determined from the system constraints. Use the algorithm in [10] to identify all the minimal path sets of the system.

Set $x_1 = x_1^u$,and set $X^* = X$,$t = 2, l = 0$ goto VI.

I) If X satisfies the condition in Theorem 7, then goto VII.

II) $x_2 = x_2 + 1$.

III) if $(x_1^l, ...) \in S$, then find x_1^{max} such that $(x_1^{max}, ...) \in S$, set $X = (x_1^{max}, ...)$, goto VI.

IV) Set $l = l + 1$, if $l > n - 2$, goto VIII.

V) Set $k = t + l, x_k = x_k + 1$, if $x_k > x_k^u$ then goto IV. Otherwise $x_j = x_j^l$ for $j = 2, ..., k - 1, l = 0$ goto III.

VI) Check for the 3-component structures in the form of $(x_a, [x_b, x_c])$ (please refer to section 5.1.1 for the meanings of "[]" and "()"), if the 3-component structure is included in all of the minimal path sets found in step (0), then swap the numbers of redundancies according to the statement in theorem 7. If this swapped redundancy assignment is feasible then we can drop out the original redundancy assignment from further consideration and return to II.

VII) Compute $R(X)$ and set $X^* = X$ if $R(X) > R(X^*)$. Goto II.

VIII) Stop.

6 On "Successive Solution Scheme For Constrained Redundancy Optimization In Reliability Networks"[1]

6.1 Introduction

We have presented in Chapter 4 and chapter 5, some special methods to improve the efficiency of problem-solving in constrained redundancy optimization problems for series-parallel systems. These methods basically employ the enumeration techniques. They are more efficient than the earlier searching methods as they will only search through a small sub-set of the whole feasible solution set, but they work only for the series-parallel systems. Therefore, we present below another novel methodology developed by Li [1], which can be used to attacked the general networks (including complex networks structures like the bridge networks.)

6.2 The contents

6.2.1 The motivation

Mathemtical schemes like the dynamic programming scheme is very powerful in solving the optimization problems that bear separable structure. But the

system reliability functions $R(X)$'s in redundancy optimization problems are often unseparable nonlinear functions. We can not apply these mathematical schemes directly to solve them in most cases.

In order to tackle with this, Li [1] proposed a new solution scheme that first converts the original $R(X)$ into an equivalent problem with concave objective function, and then transforms the problem with the concavified objective function into a separable auxiliary parametric problem. The original problem is finally solved through a separable auxiliary parametric problem.

The process of concavification ensures that the global optimal solution can be reached, while setting up the separable auxiliary parametric problem, on the other hand, enables us to find the optimal solution with the dynamic programming method or other efficient methods that require separable problem structure.

6.2.2 The Successive Solution Scheme

Consider a constrained redundancy optimization problem of the following form for general complex networks. (this part is partly cited from [1])

$$
\begin{aligned}
(P) \quad & max\, R = \phi[R_1(n_1), R_2(n_2), ..., R_k(n_k)] \\
s.t. \quad & \sum_{i=1}^{k} C_i(n_i) \leq C \\
& R_i(n_i) = 1 - (1 - r_i)^{n_i} \qquad i = 1, 2..., k \\
& 1 \leq L_i \leq n_i \leq U_i
\end{aligned}
$$

where R is the overall system reliability, R_i's are the reliabilities of each of the i^{th} subsystems, n_i's are integers that represent the number of redundancy in parallel in subsystem i, $r_i \in [0, 1]$ is the reliability of the component in subsystem i, $C_i(n_i)$'s are the resouces constraints, and C is the total resources available to the system.

Denote the redundancy assignment vector by $n = [n_1, n_2, ..., n_k]'$. The feasible region of n is given by $N = \{n | n\ satisfies\ the\ constraints\ in\ (P)\}$. The Set of optimal solution to problem (P) is defined by $N^* = \{n | n \in N\ and\ n\ is\ a\ maximizer\ in\ (P)\}$. Now Let's consider the following monotonic increasing transformation of R,

$$
\begin{aligned}
\hat{R} \quad & = \hat{\phi}[exp(rR_1(n_1), rR_2(n_2), ..., rR_k(n_k)] \\
& = -exp[\frac{r}{\phi\{\frac{1}{r} ln\, exp(rR_1), \frac{1}{r} ln\, exp(rR_2), ..., \frac{1}{r} ln\, exp(rR_k)\}}] \\
& = -exp[\frac{r}{\phi(R_1, R_2, ..., R_k)}]
\end{aligned}
$$

where r is a positive real number.

Denote by H the Hessian matrix of R with respect to $R_1, R_2, ..., R_k$, and $\hat{H}$ the Hessian matrix of $\hat{R}$ with respect to $exp(rR_1), exp(rR_2), ..., exp(rR_k)$. It is shown in [1], that $\hat{H}$ can be represented as $\hat{H} = -exp(\frac{r}{R})\Lambda[C + \frac{1}{r}(D - \frac{H}{R^2})]\Lambda$ Where R is the system reliability function, Λ is a $k \times k$ nonsingular diagonal matrix whose (i,i)th element is $\frac{1}{exp(rR_1)}$, C is also a $k \times k$ matrix given by

$$C = \begin{bmatrix} a_1(a_1+1) & a_1a_2 & ... & a_1a_i & ... & a_1a_k \\ a_2a_1 & a_2(a_2+1) & ... & a_2a_i & ... & a_2a_k \\ ... & ... & ... & ... & ... & ... \\ a_ia_1 & a_ia_2 & ... & a_i(a_i+1) & ... & a_ia_k \\ ... & ... & ... & ... & ... & ... \\ a_ka_1 & a_ka_2 & ... & a_ka_i & ... & a_k(a_k+1) \end{bmatrix}$$

with $a_i = \frac{1}{R^2}\frac{\partial R}{\partial R_i}$, $i = 1, 2, ..., k$.

D is also a $k \times k$ diagonal matrix whose (i,i)th component is $\frac{2}{R^3}(\frac{\partial R}{\partial R_i})^2$. It is proved in [1] that we can make the Hessian matrix $\hat{H}$ negative definite by selecting a r that is larger than a finite number q. We can find the smallest

q by solving the following non-linear programming problem,

$$f = minX'\{C + \frac{1}{q}[D - \frac{H}{R^2}]\}X \geq 0$$

$$s.t. R = \phi(R_1, R_2, ..., R_k)$$

$$0 \leq R_i \leq 1,\ i = 1, 2, ..., k$$

$$\| X \| = 1$$

we can then find the r by adding one to this q (i.e. $r = q + 1$). With this r, we can now transformed problem (P) to problem (E) as shown below.

$$(E) \quad max\hat{R} = \hat{\phi}[exp(rR_1(n_1), rR_2(n_2), ..., rR_k(n_k)]$$

$$s.t. \quad \sum_{i=1}^{k} C_i(n_i) \leq C$$

$$R_i(n_i) = 1 - (1 - r_i)^{n_i} \qquad i = 1, 2..., k$$

$$1 \leq L_i \leq n_i \leq U_i$$

Problem (P) and (E) are equivalent for any $r > 0$. Problem (E) is still nonlinear and nonseparable, but it is concavified. Problem (E) is now transformed into a linear and separable problem, as shown below:

$$(A(n^*)) \quad max \sum_{i=1}^{k} d_i(n^*)exp[rR_i(n_i)]$$

$$s.t. \quad constraints \quad in \quad (P)$$

where $d_i(n) = \nabla\hat{R} \mid_n$

We can search the optimal solution of (P) through the auxiliary linear and separable problem $(A(n^*))$. But we need a stopping condition that tells us

when the optimal solution of $(P)/(E)$ is achieved. It was shown in [1] that if $n^* \in N^*(n*)$ then $n^* \in N^*$. Where n^* is the optimal redundancy assignment, $N^*(n^*)$ denotes the set of optimal solutions for problem $(A(n^*))$, and N^* denotes the set of optimal solutions to the problem (P). This condition actually means that if n^* is the optimal solution of the problem (P), then we should be able to reach n^* again by starting over again at n^* in problem $(A(n^*))$. This condition can be called "stationary condition" .

The stationary condition is only a sufficient condition for the optimal solution in problem (P), it is not a necessary condition. That means not all the optimal solutions will bear this characteristics, since not all of them will have a supporting hyperplane at n^* on the *noninferior frontier* in the $\{R_1^p, R_2^p, ..., R_k^p\}$ space. The redundancy assignments on the *noninferior frontier* are assignments that are close enough to the boundary of the feasible region such that not a single extra redundancy can be added to any of the components.

But we can reshape the original noninferior frontier to the noninferior frontier that has a supporting hyperplane at n^* by keep on ignoring the solutions that are not the best in terms of the overall system reliability R up to the current iteration and put them into a black list which precludes them from further considerations.

6.3 Illustrative examples

We now need to illustrate how the methodology works. Let's first consider a simple example, then we will solve a more complicated example we found in Shi's paper[9].

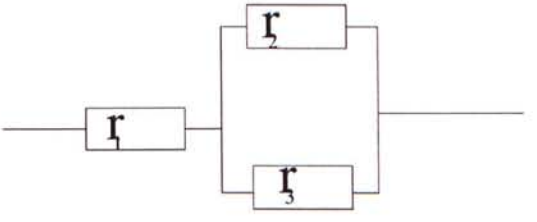


Figure 8: Example 1.

6.3.1 Example 1

For the simple network as shown above in Fig.8, we have its problem formulation as:

$$
\begin{aligned}
(P) \quad & maxR = R_1(R_2 + R_3 - R_2R_3) \\
s.t. \quad & \sum_{i=1}^{k} C_i \times n_i \leq 15 \\
& R_i(n_i) = 1 - (1 - r_i)^{n_i} \qquad i = 1, 2, 3
\end{aligned}
$$

$$where \quad C_1 = 2\,, C_2 = 3\,, C_3 = 4 \;\; 1 \leq n_i \leq 3$$

$$and \quad r_1 = 0.8\,, r_2 = 0.7\,, r_3 = 0.9$$

First we need to find a positive integer r such that $\hat{R} = -R^{-r}$ is concave. In other words we need to have the Hessian matrix of $\hat{R}$ to be negatively definite. Denote the Hessian matrix of $\hat{R}$ by $\hat{H}$, we then have:

$$\hat{H} = -exp(\frac{r}{R})\Lambda[C + \frac{1}{r}(D - \frac{H}{R^2})]$$

$where \quad R = R_1(R_2 + R_3 - R_2R_3)$

$\Lambda\, is\, a\, 3 \times 3\, matrix\, with\, its\, (i,i)th\, elements\, being\, \dfrac{1}{exp(rRi)}$

$$C = \begin{pmatrix} a_1(a_1+1) & a_1a_2 & a_1a_3 \\ a_2a_1 & a_2(a_2+1) & a_2a_3 \\ a_3a_1 & a_3a_2 & a_3(a_3+1) \end{pmatrix}$$

$$a_i = \frac{1}{R^2}\frac{\partial R}{\partial R_i}$$

$D\, is\, a\, 3 \times 3\, matrix\, with\, its\, (i,i)th\, elements\, being\, \dfrac{2}{R^3}(\dfrac{\partial R}{\partial R_i})^2$

$$H = \begin{pmatrix} 0 & 1-R_3 & 1-R_2 \\ 1-R_3 & 0 & -R_1 \\ 1-R_2 & -R_1 & 0 \end{pmatrix}$$

We know that if $\hat{H}$ is to be negatively definite then $\Lambda[C + \frac{1}{r}(D - \frac{H}{R^2})]$

need to be positive definite. This is equivalent to solving for r such that:

$$
\begin{aligned}
min \quad & \{X'[C + \frac{1}{r}(D - \frac{H}{R^2}]X\} \geq 0 \\
& x_1^2 + x_2^2 + x_3^2 = 1 \\
& r_1 \leq R_1 \leq 1 - (1 - r_1)^3 \\
& r_2 \leq R_2 \leq 1 - (1 - r_2)^3 \\
& r_3 \leq R_3 \leq 1 - (1 - r_3)^3
\end{aligned}
$$

We solved the above non-linear programming problem by using a software package called "GINO" [19], and the value for the smallest r to concavify the problem is 20. With this value of r, the Hessian matrix $\hat{H}$ will be negative definite. That means the original $R(X)$ can be concavified into $\hat{R}(X)$ by this r.

Now we need to transform the original problem into a separable problem $(A(n^*))$. That means we need to substitute the corresponding values into $(A(n^*))$. Let's first assume n^*=(1,1,1), or in other words let's assume that the optimal redundancy assignment for this problem is 1 for all of the components 1, 2 and 3. Of course this point can't be the optimal solution since it is not even a noninferior point, we choose it because we need a place to start with. We substitute the values into to the apporiate places and find that $d(n) = [28281, 17235, 947]$(correct to the nearest integer). Now we have the

vector to start with our problem, we used dynamic programming to solve this problem.

Iteration 1:

with the vector $d(n) = [28281, 17235, 947]$ we find the largest value for $(A(n^*))$ is 1.17829×10^{13} at (3,1,1), that means the optimal solution for $(A(n^*))$ from (1,1,1) is at (3,1,1). Since the system reliability at (3,1,1) (i.e. R(3,1,1)) is better than at (1,1,1) (i.e. R(1,1,1)), therefore (1,1,1) is put to the black list and will be ignored. n^* now becomes (3,1,1).

Black list: {(1,1,1)}

Iteration 2:

We need to start over again from (3,1,1). The new $d(n)$ is $[2.695, 94.746, 5.206]$. We start from this point and find that largest value of $(A(n^*))$ is $27, 153, 231, 778$ at (1,3,1). But the system reliabilty at (1,3,1) is lower than the system reliability at (3,1,1), (i.e. $R(1,3,1) < R(3,1,1)$). We need to remain at (3,1,1) and put the point (1,3,1) into our black list.

Black list: {(1,1,1),(1,3,1)}

Iteration 3:

We stay at (3,1,1), the $d(n)$ is still the same as the one in iteration 2. And after ignoring the points inside the black list, we find that the largest value

of $(A(n^*))$ is $8,527,702,794$ at (2,2,1). The system reliability of R(2,2,1) is however still less than R(3,1,1). Therefore we need to continue remain at (3,1,1) and (2,2,1) will become a new member in our black list set.

Black list: {(1,1,1),(1,3,1),(2,2,1)}

Iteration 4:

We still stay at (3,1,1). Ignoring all the black-listed points will give us the $2,769,374,224$ at (2,1,2) as the largest value for $(A(n^*))$. But unfortunately the system reliability of R(2,1,2) is also less than R(3,1,1). We have no choice but to put this point into our black list and stay at the original point (3,1,1).

Black list: {(1,1,1),(1,3,1),(2,2,1),(2,1,2)}

Iteration 5:

We start from (3,1,1). Again we need to ignore all the black-listed points, and we find that the largest value is at (3,1,1). Recall from the stationary condition mentioned in section 6.2.2, we find that we have reached (3,1,1) again from the same point (3,1,1), this matches with the stationary condition! Since the stationary condition is a sufficient condition for optimality, therefore the point (3,1,1) is the optimal solution of problem $(P)/(E)$. This completes our solution finding iterations. The optimal redundancy assign-

ment for our example 1 is at (3,1,1), which means we should assign 3 redundancies to component 1, and 1 redundancy to components 2 and 3. Later computational results show that (3,1,1) is indeed the optimal solution for the problem (P)

.

6.3.2 Example 2

We have seen how the successive methodology works with a small example. we need to try it with a more complicated example in the literature [9] .

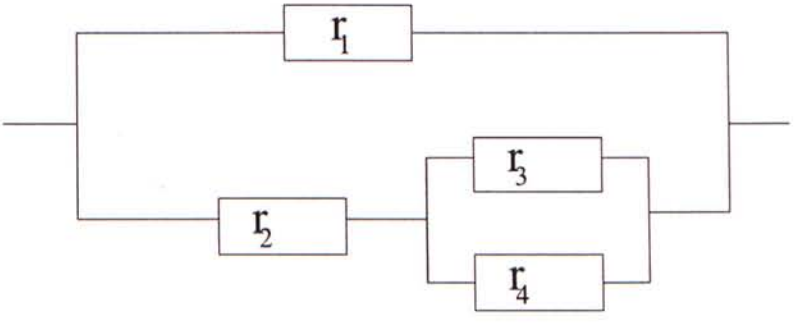


Figure 9: Example 2.

For the network shown in Fig.9, we can formulate it as follows:

$$(P) \quad max\, R = R_1 + R_2(R_3 + R_4 - R_3R_4) - R_1R_2(R_3 + R_4 - R_3R_4)$$

$$s.t. \quad \sum_{i=1}^{k} C_i \times n_i \leq 30$$

$$\sum_{i=1}^{k} W_i \times n_i \leq 40$$

$$R_i(n_i) = 1 - (1 - r_i)^{n_i} \qquad i = 1, 2, 3, 4$$

where $C_1 = 6\,, C_2 = 4\,, C_3 = 3\,, C_4 = 2$

$W_1 = 9\,, W_2 = 4\,, W_3 = 4\,, W_4 = 3$

and $r_1 = 0.80\,, r_2 = 0.75\,, r_3 = 0.70\,, r_4 = 0.65$

Again we need to find the integer r such that $\hat{R}$=-R^r is concave. Same as before, we need the Hessian matrix $\hat{H}$ of $\hat{R}$ to be negatively definite. The Hessian matrix $\hat{H}$ is similar to the previous one and was listed below:

$$\hat{H} = -exp(\frac{r}{R})\Lambda[C + \frac{1}{r}(D - \frac{H}{R^2})]$$

But the R, Λ,C, D and H are a little bit different from their previous counterparts and they are listed as follows.

$$R = R_1 + R_2(R_3 + R_4 - R_3R_4) - R_1R_2(R_3 + R_4 - R_3R_4)$$

$$\Lambda \, is \, a \, 4 \times 4 \, matrix \, with \, its \, (i,i)th \, elements \, being \, \frac{1}{exp(rRi)}$$

$$C = \begin{pmatrix} a_1(a_1+1) & a_1a_2 & a_1a_3 & a_1a_4 \\ a_2a_1 & a_2(a_2+1) & a_2a_3 & a_2a_4 \\ a_3a_1 & a_3a_2 & a_3(a_3+1) & a_3a_4 \\ a_4a_1 & a_4a_2 & a_4(a_3) & a_4(a_4+1) \end{pmatrix}$$

$$a_i = \frac{1}{R^2}\frac{\partial R}{\partial R_i}$$

$$D \, is \, a \, 4 \times 4 \, matrix \, with \, its \, (i,i)th \, elements \, being \, \frac{2}{R^3}(\frac{\partial R}{\partial R_i})^2$$

$$H = \begin{pmatrix} 0 & -R_3 - R_4 + R_3R_4 & -R_2 + R_2R_4 & -R_2 + R_2R_3 \\ -R_3 - R_4 + R_3R_4 & 0 & 1 - R_1 - R_4 + R_1R_4 & 1 - R_1 - R_3 + R_1R_3 \\ -R_2 + R_2R_4 & 1 - R_1 - R_4 + R_1R_4 & 0 & -R2 + R_1R_2 \\ -R_2 + R_2R_3 & 1 - R_1 - R_3 + R_1R_3 & -R2 + R_1R_2 & 0 \end{pmatrix}$$

We still use GINO [19] to solve for the r, and we find that if r is 1537 then the Hessian matrix $\hat{H}$ will be negatively definite.

We now continue to solve for the new $(A(n^*))$. Same as before we still use dynamic programming to solve it, and we start from (1,1,1,1) this time, the $d(n)$ at this point is $[exp(414.46), exp(490.70), exp(566.32), exp(645.56)]$

Iteration 1: We use dynamic programming to find the solution for $(A(n^*))$ from $d(n)$, and we arrive at (1,1,3,4). The system reliability R(1,1,3,4) is larger than R(1,1,1,1), therefore (1,1,1,1) is put to the black list and will

be ignored. n^* now becomes (1,1,3,4).

Black list: {(1,1,1,1)}

Iteration 2: We start over at (1,1,3,4). The $d(n)$ now becomes $[exp(387.01), exp(463.64), exp(116.40), exp(98.55)]$. With this new $d(n)$ we arrive at (1,4,2,1). The system reliability R(1,4,2,1) is better than R(1,1,3,4), therefore (1,1,3,4) is put to the black list. n^* then becomes (1,4,2,1).

Black list: {(1,1,1,1),(1,1,3,4)}

Iteration 3: We now start from (1,4,2,1). The $d(n)$ becomes $[exp(314.99), exp(15.30), exp(146.60), exp(544.86)]$. With this $d(n)$ we stop at (2,1,2,4). But the system reliabilty R(2,1,2,4) is worse than R(1,4,2,1), therefore we need to put (2,1,2,4) into the black list and start over from (1,4,2,1) again.

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4)}

Iteration 4: We start again from (1,4,2,1). We stop at (2,1,1,4). But R(2,1,1,4) is lower than R(1,4,2,1), therefore (2,1,1,4) is put into the black list and we need to still start from (1,4,2,1).

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4)}

Iteration 5: We start from (1,4,2,1). Then we stop at (1,2,2,4). R(1,2,2,4) is also lower that R(1,4,2,1), as a result (1,2,2,4) is put into the black list. We have to start again from (1,4,2,1).

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4)}

Iteration 6: We start from (1,4,2,1). And finally we arrive at (1,3,1,4). R(1,3,1,4) is better than R(1,4,2,1). So (1,4,2,1) is put into the black list. We need now starting from (1,3,1,4).

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4),(1,4,2,1)}

Iteration 7: We now start from (1,3,1,4). We arrive at (2,1,4,1). R(2,1,4,1) is worse than R(1,3,1,4). (2,1,4,1) is therefore put into the black list. We have to start again from (1,3,1,4) in the next iteration.

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4),(1,4,2,1),(2,1,4,1)}

Iteration 8: We still start from (1,3,1,4). We arrive at (1,1,4,4). R(1,1,4,4)is worse than R(1,3,1,4). (1,1,4,4) is put into the black list. We start again from (1,3,1,4).

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4),(1,4,2,1),(2,1,4,1), (1,1,4,4)}

Iteration 9: We start from (1,3,1,4). We now arrive at (1,2,4,2). R(1,2,4,2) is no better than R(1,3,1,4), therefore we put (1,2,4,2) into the black list.

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4),(1,4,2,1),(2,1,4,1), (1,1,4,4),(1,2,4,2)}

Iteration 10: We still start from (1,3,1,4). We arrive at (2,1,3,2). R(2,1,3,2) is not as good as R(1,3,1,4). So (2,1,3,2) is put into the black list.

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4),(1,4,2,1),(2,1,4,1), (1,1,4,4),(1,2,4,2),(2,1,3,2)}

Iteration 11: We start from (1,3,1,4). We arrive at (1,3,3,1). R(1,3,3,1) is still no better than R(1,3,1,4). (1,3,3,1) is put into the black list.

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4),(1,4,2,1),(2,1,4,1), (1,1,4,4),(1,2,4,2),(2,1,3,2),(1,3,3,1)}

Iteration 12: We start from (1,3,1,4). We arrive at (1,2,3,3). R(1,2,3,3) is smaller than R(1,3,1,4). (1,2,3,3,) is put into the black list.

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4),(1,4,2,1),(2,1,4,1), (1,1,4,4),(1,2,4,2),(2,1,3,2),(1,3,3,1),(1,2,3,3)}

Iteration 13: We start from (1,3,1,4). Finally we arrive at (2,2,2,2). R(2,2,2,2) is better than R(1,3,1,4). Therefore (1,3,1,4) is now put into the black list.

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4),(1,4,2,1),(2,1,4,1), (1,1,4,4),(1,2,4,2),(2,1,3,2),(1,3,3,1),(1,2,3,3),(1,3,1,4)}

Iteration 14: We now start from (2,2,2,2). We arrive at (1,4,1,3). R(1,4,1,3) is worse than R(2,2,2,2). (1,4,1,3) is put into the black list.

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4),(1,4,2,1),(2,1,4,1), (1,1,4,4),(1,2,4,2),(2,1,3,2),(1,3,3,1),(1,2,3,3),(1,4,1,3)}

Iteration 15: We start from (2,2,2,2). We arrive at (1,3,2,3). R(1,3,2,3) is worse than R(2,2,2,2). (1,3,2,3) is put into the black list.

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4),(1,4,2,1),(2,1,4,1), (1,1,4,4),(1,2,4,2),(2,1,3,2),(1,3,3,1),(1,2,3,3),(1,4,1,3),(1,3,2,3)}

Iteration 16: We start from (2,2,2,2). We arrive at (2,2,1,3). R(2,2,1,3) is worse than R(2,2,2,2). (2,2,1,3) is put into the black list(note that (2,2,1,3) is the optimal solution found in [9]).

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4),(1,4,2,1),(2,1,4,1), (1,1,4,4),(1,2,4,2),(2,1,3,2),(1,3,3,1),(1,2,3,3),(1,4,1,3),(1,3,2,3),(2,2,1,3)}

Iteration 17: We start from (2,2,2,2). We arrive at (1,2,4,2). R(1,2,4,2) is not as good as R(2,2,2,2). (1,2,4,2) is put into the black list.

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4),(1,4,2,1),(2,1,4,1), (1,1,4,4),(1,2,4,2),(2,1,3,2),(1,3,3,1),(1,2,3,3),(1,4,1,3),(1,3,2,3),(2,2,1,3),(1,2,4,2)}

Iteration 18: We start from (2,2,2,2) again. We arrive at (3,1,1,1). R(3,1,1,1) is better than R(2,2,2,2). (2,2,2,2) is put into the black list.

Black list: {(1,1,1,1),(1,1,3,4),(2,1,2,4),(2,1,1,4),(1,2,2,4),(1,4,2,1),(2,1,4,1), (1,1,4,4),(1,2,4,2),(2,1,3,2),(1,3,3,1),(1,2,3,3),(1,4,1,3),(1,3,2,3),(2,2,1,3),(1,2,4,2) ,(2,2,2,2)}

last iterations: We start from (3,1,1,1) and we eventually arrived at (3,1,1,1). (3,1,1,1) is therefore the best solution since it satisfies the stationary condition. Later computation results reveal that (3,1,1,1) is really the optimal solution to $(P)/(E)$ with an overall system reliability of 0.99737.

We note that in Shi's paper[9] the optimal solution is 0.9970 at (2,2,1,3) but our calculations show that the global optimal solution is 0.997375 at (3,1,1,1). This show the advantage of the exact solution scheme over the heuristic search solution schemes.

The new solution scheme can be very efficient while solving small-scale single-constraint problems. It will, however, be hindered if there exists no supporting hyper-plane on the noninferior frontier in the $R_1^p, R_2^p, ..., R_k^p$ space for the optimal solution. We have seen in our examples that extra iterations are needed to ensure the optimality even though the optimal solutions have already been reached. It is theoretical explorable. But the problem examples show it is somehow a bit inefficient if the optimal solution does not locate in a "nice" place.

7 Conclusions

In this thesis we have presented two approaches to deal with the constrained redundancy optimization problems.

The first approach is to solve the problem through an implicit search. The efficiency of this type of approach mainly depends on the size of the search space. Some major efforts have been put to reduce the search space. We have successfully reduced the search space for the series-parallel networks by considering the differences between element redundant levels and this is described in Chapter 4. We have also developed another efficient solution scheme for certain types of series-parallel reducible networks in Chapter 5. Further refinery to it still seems possible by considering the differences between the redundant levels.

These two solution schemes are indirect solution schemes, there are some limitations. First, if we wish to confine the search space even further, we will need to gather more information and derive more knowledge. This kind of knowledge, can become increasingly expensive to get, in terms of computational time and memory. Second, we are almost blind in choosing the appropriate direction to search. All we can do now is just to restrict the search space to be as close to the feasible frontier as possible.

One possible approach to solve the first problem is to have a comprehen-

sive investigation on the type of knowledge we need to acquire on the common and major reliability networks, and to construct certain type of look up chart (like the logrithm table) for rapid knowledge acquisitions. Problem two may be solved by developing another algorithm that is similar to the MIP [12] algorithm that can make use of the results we described in Chapter 4 and Chapter 5.

As for the second approach we presented in chapter 6, it can be viewed as a direct method of solving the constrained redundancy optimization problems. It first concavifies the objective problem and then builds another separable auxiliary problem for it. The original problem can be solved through solving the auxiliary problem. It can be very efficient if the optimal solution has a supporting hyper-plane on the noninferior frontier in the $\{R_1^p, R_2^p, ..., R_k^p\}$ space. But unfortunately this is not always the case. If in the case of "poorly" located optimal solution, this approach will require some extra iterations to be made. The black-listed points will then accumulate and the black-list can become very large. Dynamic programming which is an efficient method to solve the auxiliary problem may also run into dimensional difficulties if there are more than one resource constraints that is imposed on the problem.

In order to solve the augmenting black-list problem, we need either 1) an efficient method to identify and eliminate the black-listed points in groups. Or 2) using another approach to solve the separable auxiliary problem. We

may also need choosing another methodology to replace the dynamic programming methodology in solving multi-constraint problems so as to avoid the dimensional difficulties.

8 Appendix

Please note in the below that the symbol "o" means the point is a non-inferior point. The symbol "*" means this point is a maximal monotone point, the symbol "!" means this point is a proper maximal monotone point.

8.1 Computational results

Results for example 1:

List of Maximal monotone points

No.	X	Maximal monotone	R(X)
1	(3,1,1,1)	*	0.40199
2	(2,3,1,1)	*	0.51228
3	(1,4,2,1)	*	0.48355
4	(1,3,3,1)	*	0.49805
5	(1,1,6,1)	*	0.36391
6	(2,2,2,2)	!	0.71867
7	(1,2,4,2)	*	0.56562
8	(1,3,2,3)	!	0.58818
9	(1,2,2,5)	!	0.55435

The optimal point is at : (2,2,2,2)
The optimal reliability is: 0.71867

Results for example 2:

List of Maximal monotone points

No.	X	Maximal monotone	R(X)
1	(3,2,2,1,1)	!	0.66991
2	(5,1,1,1,2)	*	0.50367
3	(4,2,1,1,2)	*	0.62601
4	(3,3,1,1,2)	!	0.64479

```
5  (2,2,2,1,3)              !              0.69545
6  (4,1,1,1,4)              *              0.50582
7  (3,2,1,1,4)              !              0.62023
8  (3,1,1,1,6)              *              0.49623
9  (2,2,1,1,6)              !              0.58012
10  (2,1,1,1,8)              !              0.46410
11  (1,1,1,1,10)              !              0.35700
The optimal point is at    : (2,2,2,1,3)
The optimal reliability is: 0.69545
```

Results for example 3:

List of Maximal monotone points

No.	X	Maximal monotone	R(X)
1	(8,2,1,1,1)	*	0.57362
2	(7,4,1,1,1)	*	0.60922
3	(6,5,1,1,1)	*	0.61028
4	(7,2,2,1,1)	*	0.68806
5	(6,4,2,1,1)	*	0.73019
6	(6,3,3,1,1)	*	0.74565
7	(5,4,3,1,1)	*	0.75195
8	(4,4,4,1,1)	*	0.74938
9	(6,3,2,2,1)	*	0.82984
10	(5,4,2,2,1)	*	0.83684
11	(5,3,3,2,1)	*	0.85456
12	(4,3,4,2,1)	*	0.85164
13	(4,3,3,3,1)	*	0.86274
14	(7,3,1,1,2)	*	0.66225
15	(6,2,2,2,2)	*	0.86935
16	(5,3,2,2,2)	*	0.90970
17	(4,3,3,2,2)	!	0.93080

```
The optimal point is at    : (4,3,3,2,2)
 The optimal reliability is: 0.93080
```

Results for example 4a (fortran results) :

List of noninferior and efficient point

p= 110.0000 c= 175.0000 w= 200.0000

1	1	5	1	1	1	1	*	0.4566	108.00	128.95	145.80
2	2	4	4	1	1	1	*	0.6005	102.00	130.73	192.63
3	2	1	5	1	1	1	o	0.3974	60.00	97.91	194.30
4	3	4	3	3	1	1	*	0.7358	96.00	144.16	187.40
5	3	4	2	4	1	1	o	0.7053	93.00	149.50	191.57
6	3	3	3	4	1	1	o	0.7196	75.00	140.96	191.92
7	3	3	1	5	1	1	o	0.5490	68.00	142.03	192.37
8	3	2	2	5	1	1	o	0.6291	54.00	134.27	192.17
9	3	3	4	2	2	1	o	0.8052	83.00	135.43	195.70
10	3	2	4	3	2	1	o	0.7628	68.00	132.49	198.75
11	3	4	1	4	2	1	o	0.6489	93.00	153.59	189.56
12	3	2	3	4	2	1	o	0.7587	61.00	137.30	189.71
13	3	2	1	5	2	1	o	0.5788	54.00	138.37	190.15
14	3	1	2	5	2	1	o	0.5359	48.00	131.54	196.19
15	3	4	3	1	3	1	o	0.6958	104.00	144.16	192.47
16	3	2	4	1	3	1	o	0.6272	70.00	122.94	187.71
17	3	1	4	2	3	1	o	0.5575	61.00	120.21	189.72
18	3	3	1	4	3	1	o	0.6428	75.00	149.46	186.85
19	3	2	2	4	3	1	o	0.7367	61.00	141.70	186.65
20	3	4	1	2	4	1	o	0.6380	105.00	153.59	197.13
21	3	3	2	2	4	1	o	0.7749	83.00	144.64	188.12
22	3	2	3	2	4	1	o	0.7459	73.00	137.30	197.28
23	3	3	1	3	4	1	o	0.6406	82.00	149.46	191.38
24	3	2	2	3	4	1	o	0.7341	68.00	141.70	191.18
25	3	1	1	4	4	1	o	0.4378	57.00	135.17	192.63
26	3	2	1	1	5	1	o	0.4738	72.00	128.82	190.21
27	3	1	2	1	5	1	o	0.4387	66.00	121.99	196.26
28	3	1	1	2	5	1	o	0.4212	63.00	126.09	192.23
29	3	3	4	2	1	2	o	0.7702	86.00	132.70	195.70
30	3	2	4	3	1	2	o	0.7296	71.00	129.76	198.75
31	3	4	1	4	1	2	o	0.6207	96.00	150.86	189.56
32	3	2	3	4	1	2	o	0.7257	64.00	134.57	189.71
33	3	2	1	5	1	2	o	0.5536	57.00	135.64	190.15
34	3	1	2	5	1	2	o	0.5126	51.00	128.82	196.19
35	3	3	4	1	2	2	o	0.7381	89.00	132.70	197.71
36	4	4	3	2	2	2	!	0.9008	106.00	150.26	198.24
37	4	2	4	2	2	2	o	0.8120	72.00	129.04	193.48
38	4	4	2	3	2	2	o	0.8865	101.00	154.66	192.13
39	5	3	3	3	2	2	!	0.9045	83.00	146.12	192.48
40	5	3	2	4	2	2	o	0.8670	80.00	151.46	196.65

41	5	1	3	4	2	2	o	0.6182	58.00	131.84	193.73
42	5	1	1	5	2	2	o	0.4716	51.00	132.91	194.18
43	5	1	4	1	3	2	o	0.5111	67.00	117.48	191.74
44	5	4	2	2	3	2	o	0.8747	106.00	154.66	195.19
45	5	3	3	2	3	2	o	0.8924	88.00	146.12	195.53
46	5	4	1	3	3	2	o	0.7231	105.00	159.48	198.44
47	5	3	2	3	3	2	o	0.8783	83.00	150.53	189.43
48	5	2	3	3	3	2	o	0.8454	73.00	143.19	198.59
49	5	2	1	4	3	2	o	0.6483	64.00	143.07	184.64
50	5	1	2	4	3	2	o	0.6003	58.00	136.24	190.68
51	5	3	2	1	4	2	o	0.7103	89.00	141.91	190.14
52	5	2	3	1	4	2	o	0.6838	79.00	134.57	199.30
53	5	3	1	2	4	2	o	0.6819	86.00	146.00	186.11
54	5	2	2	2	4	2	o	0.7815	72.00	138.25	185.91
55	5	2	1	3	4	2	o	0.6460	71.00	143.07	189.16
56	5	1	2	3	4	2	o	0.5982	65.00	136.24	195.20
57	5	1	1	1	5	2	o	0.3861	69.00	123.36	194.24
58	5	2	4	1	1	3	o	0.5938	78.00	117.27	187.71
59	5	4	2	2	1	3	o	0.7528	105.00	142.17	179.08
60	5	1	4	2	1	3	o	0.5278	69.00	114.54	189.72
61	5	3	1	4	1	3	o	0.6086	83.00	143.79	186.85
62	5	2	2	4	1	3	o	0.6974	69.00	136.03	186.65
63	5	4	2	1	2	3	o	0.7214	108.00	142.17	181.09
64	5	1	4	1	2	3	o	0.5058	72.00	114.54	191.74
65	5	3	3	2	2	3	o	0.8832	93.00	143.19	195.53
66	5	4	1	3	2	3	o	0.7156	110.00	156.55	198.44
67	5	3	2	3	2	3	o	0.8692	88.00	147.59	189.43
68	5	2	3	3	2	3	o	0.8368	78.00	140.25	198.59
69	5	2	1	4	2	3	o	0.6416	69.00	140.13	184.64
70	5	1	2	4	2	3	o	0.5941	63.00	133.31	190.68
71	5	2	3	1	3	3	o	0.6880	80.00	130.70	187.55
72	5	3	2	2	3	3	o	0.8576	93.00	147.59	192.48
73	5	1	3	2	3	3	o	0.6116	71.00	127.97	189.56
74	5	3	1	3	3	3	o	0.7090	92.00	152.41	195.74
75	5	2	2	3	3	3	o	0.8125	78.00	144.66	195.53
76	5	1	1	4	3	3	o	0.4846	67.00	138.12	196.99
77	5	3	1	1	4	3	o	0.5734	98.00	143.79	196.44
78	5	2	2	1	4	3	o	0.6571	84.00	136.03	196.24
79	5	2	1	2	4	3	o	0.6308	81.00	140.13	192.22
80	5	1	2	2	4	3	o	0.5841	75.00	133.31	198.26
81	5	3	2	2	1	4	o	0.7321	98.00	135.77	188.12

82	5	2	3	2	1	4	o	0.7048	88.00	128.43	197.28
83	5	3	1	3	1	4	o	0.6052	97.00	140.59	191.38
84	5	2	2	3	1	4	o	0.6936	83.00	132.83	191.18
85	5	1	1	4	1	4	o	0.4137	72.00	126.30	192.63
86	5	3	2	1	2	4	o	0.7016	101.00	135.77	190.14
87	5	2	3	1	2	4	o	0.6754	91.00	128.43	199.30
88	5	3	1	2	2	4	o	0.6736	98.00	139.86	186.11
89	5	2	2	2	2	4	o	0.7719	84.00	132.11	185.91
90	5	2	1	3	2	4	o	0.6381	83.00	136.93	189.16
91	5	1	2	3	2	4	o	0.5908	77.00	130.10	195.20
92	5	3	1	1	3	4	o	0.5723	105.00	140.59	196.44
93	5	2	2	1	3	4	o	0.6558	91.00	132.83	196.24
94	5	2	1	2	3	4	o	0.6296	88.00	136.93	192.22
95	5	1	2	2	3	4	o	0.5830	82.00	130.10	198.26
96	5	2	1	1	1	5	o	0.4475	96.00	116.40	190.21
97	5	1	2	1	1	5	o	0.4144	90.00	109.58	196.26
98	5	1	1	2	1	5	o	0.3978	87.00	113.67	192.23
99	5	1	1	1	2	5	o	0.3812	90.00	113.67	194.24

n_f= 494 n_n= 99 5 0.0101 0.0505

Results for example 4b:

List of noninferior and efficient point

p= 114.0000 c= 185.0000 w= 212.0000

1	1	5	2	1	1	1	*	0.5707	114.00	134.40	163.92
2	1	2	5	1	1	1	o	0.5365	72.00	110.20	206.38
3	1	5	1	2	1	1	o	0.5479	111.00	138.50	159.89
4	2	4	4	2	1	1	*	0.7206	105.00	140.29	206.72
5	2	1	5	2	1	1	o	0.4769	63.00	107.47	208.40
6	2	3	4	3	1	1	o	0.7235	82.00	136.15	200.97
7	2	1	4	4	1	1	o	0.4945	57.00	121.87	202.22
8	2	3	2	5	1	1	o	0.6862	74.00	147.49	210.49
9	2	1	3	5	1	1	o	0.4893	52.00	127.86	207.57
10	2	5	1	1	2	1	o	0.5251	114.00	138.50	161.91
11	2	4	4	1	2	1	o	0.6905	108.00	140.29	208.74
12	2	1	5	1	2	1	o	0.4570	66.00	107.47	210.41
13	3	4	3	3	2	1	*	0.8462	102.00	153.71	203.51
14	3	2	4	3	2	1	o	0.7628	68.00	132.49	198.75
15	3	4	2	4	2	1	o	0.8111	99.00	159.05	207.68
16	3	3	3	4	2	1	o	0.8275	81.00	150.51	208.03

17	3	3	1	5	2	1	o	0.6313	74.00	151.58	208.48
18	3	2	2	5	2	1	o	0.7235	60.00	143.83	208.28
19	3	3	4	1	3	1	o	0.6841	90.00	136.15	206.03
20	3	4	3	2	3	1	o	0.8349	107.00	153.71	206.56
21	3	2	4	2	3	1	o	0.7526	73.00	132.49	201.81
22	3	4	2	3	3	1	o	0.8217	102.00	158.12	200.45
23	4	3	3	3	3	1	*	0.8383	84.00	149.58	200.80
24	4	1	4	3	3	1	o	0.5761	66.00	130.49	211.10
25	4	3	2	4	3	1	o	0.8036	81.00	154.91	204.98
26	4	1	3	4	3	1	o	0.5730	59.00	135.29	202.06
27	4	1	1	5	3	1	o	0.4371	52.00	136.36	202.50
28	4	4	2	1	4	1	o	0.6645	108.00	149.50	201.16
29	4	3	3	1	4	1	o	0.6780	90.00	140.96	201.51
30	4	1	4	1	4	1	o	0.4659	72.00	121.87	211.81
31	4	4	1	2	4	1	o	0.6380	105.00	153.59	197.13
32	4	2	3	2	4	1	o	0.7459	73.00	137.30	197.28
33	4	3	2	3	4	1	o	0.8007	88.00	154.91	209.50
34	4	1	3	3	4	1	o	0.5710	66.00	135.29	206.58
35	4	2	1	4	4	1	o	0.5911	69.00	147.45	204.71
36	4	1	2	4	4	1	o	0.5473	63.00	140.63	210.75
37	4	3	1	1	5	1	o	0.5168	92.00	142.03	208.54
38	4	2	2	1	5	1	o	0.5923	78.00	134.27	208.34
39	4	2	1	2	5	1	o	0.5686	75.00	138.37	204.31
40	4	1	2	2	5	1	o	0.5265	69.00	131.54	210.35
41	4	4	4	1	1	2	o	0.6605	111.00	137.56	208.74
42	4	1	5	1	1	2	o	0.4372	69.00	104.74	210.41
43	4	4	3	3	1	2	o	0.8094	105.00	150.98	203.51
44	4	2	4	3	1	2	o	0.7296	71.00	129.76	198.75
45	4	4	2	4	1	2	o	0.7758	102.00	156.32	207.68
46	4	3	3	4	1	2	o	0.7916	84.00	147.78	208.03
47	4	3	1	5	1	2	o	0.6039	77.00	148.85	208.48
48	4	2	2	5	1	2	o	0.6920	63.00	141.10	208.28
49	5	4	3	2	2	2	!	0.9008	106.00	150.26	198.24
50	5	3	4	2	2	2	o	0.8857	92.00	142.25	211.81
51	5	4	2	3	2	2	o	0.8865	101.00	154.66	192.13
52	6	3	3	3	2	2	!	0.9045	83.00	146.12	192.48
53	6	1	4	3	2	2	o	0.6216	65.00	127.03	202.78
54	6	4	1	4	2	2	o	0.7138	102.00	160.41	205.67
55	6	3	2	4	2	2	o	0.8670	80.00	151.46	196.65
56	6	2	3	4	2	2	o	0.8346	70.00	144.12	205.81
57	6	2	1	5	2	2	o	0.6367	63.00	145.19	206.26

58	6	4	3	1	3	2	o	0.7653	113.00	150.98	208.57
59	6	2	4	1	3	2	o	0.6899	79.00	129.76	203.82
60	6	4	2	2	3	2	o	0.8747	106.00	154.66	195.19
61	6	3	3	2	3	2	o	0.8924	88.00	146.12	195.53
62	6	1	4	2	3	2	o	0.6133	70.00	127.03	205.83
63	6	4	1	3	3	2	o	0.7231	105.00	159.48	198.44
64	6	3	2	3	3	2	o	0.8783	83.00	150.53	189.43
65	6	2	3	3	3	2	o	0.8454	73.00	143.19	198.59
66	6	3	1	4	3	2	o	0.7071	84.00	156.28	202.96
67	6	2	2	4	3	2	o	0.8104	70.00	148.52	202.76
68	6	4	1	1	4	2	o	0.5848	111.00	150.86	199.15
69	6	2	3	1	4	2	o	0.6838	79.00	134.57	199.30
70	6	3	2	2	4	2	o	0.8524	92.00	151.46	204.23
71	6	1	3	2	4	2	o	0.6078	70.00	131.84	201.31
72	6	3	1	3	4	2	o	0.7046	91.00	156.28	207.48
73	6	2	2	3	4	2	o	0.8075	77.00	148.52	207.28
74	6	1	1	4	4	2	o	0.4816	66.00	141.99	208.74
75	6	2	1	1	5	2	o	0.5212	81.00	135.64	206.32
76	6	1	1	2	5	2	o	0.4633	72.00	132.91	208.34
77	6	4	3	1	1	3	o	0.6587	112.00	138.49	192.47
78	6	3	4	1	1	3	o	0.6477	98.00	130.49	206.03
79	6	2	4	2	1	3	o	0.7125	81.00	126.83	201.81
80	6	4	2	3	1	3	o	0.7779	110.00	152.45	200.45
81	6	3	3	3	1	3	o	0.7936	92.00	143.91	200.80
82	6	1	4	3	1	3	o	0.5454	74.00	124.82	211.10
83	6	3	2	4	1	3	o	0.7607	89.00	149.25	204.98
84	6	1	3	4	1	3	o	0.5425	67.00	129.63	202.06
85	6	1	1	5	1	3	o	0.4138	60.00	130.70	202.50
86	6	2	4	1	2	3	o	0.6828	84.00	126.83	203.82
87	6	4	2	2	2	3	o	0.8657	111.00	151.73	195.19
88	6	3	3	2	2	3	o	0.8832	93.00	143.19	195.53
89	6	1	4	2	2	3	o	0.6070	75.00	124.10	205.83
90	6	4	1	3	2	3	o	0.7156	110.00	156.55	198.44
91	6	3	2	3	2	3	o	0.8692	88.00	147.59	189.43
92	6	2	3	3	2	3	o	0.8368	78.00	140.25	198.59
93	6	3	1	4	2	3	o	0.6999	89.00	153.34	202.96
94	6	2	2	4	2	3	o	0.8021	75.00	145.59	202.76
95	6	4	1	1	3	3	o	0.5884	112.00	146.99	187.40
96	6	3	3	1	3	3	o	0.7504	100.00	143.91	205.87
97	6	3	2	2	3	3	o	0.8576	93.00	147.59	192.48
98	6	2	3	2	3	3	o	0.8256	83.00	140.25	201.64

99	6	3	1	3	3	3	o	0.7090	92.00	152.41	195.74
100	6	2	2	3	3	3	o	0.8125	78.00	144.66	195.53
101	6	1	3	3	3	3	o	0.6320	76.00	138.25	210.94
102	6	2	1	4	3	3	o	0.6542	79.00	150.41	209.07
103	6	1	3	1	4	3	o	0.5111	82.00	129.63	211.64
104	6	3	1	2	4	3	o	0.6881	101.00	153.34	210.54
105	6	2	2	2	4	3	o	0.7886	87.00	145.59	210.34
106	6	1	1	3	4	3	o	0.4829	74.00	138.12	201.51
107	6	3	3	1	1	4	o	0.6406	105.00	132.09	201.51
108	6	1	4	1	1	4	o	0.4402	87.00	113.00	211.81
109	6	2	3	2	1	4	o	0.7048	88.00	128.43	197.28
110	6	3	2	3	1	4	o	0.7565	103.00	146.05	209.50
111	6	1	3	3	1	4	o	0.5395	81.00	126.42	206.58
112	6	2	1	4	1	4	o	0.5585	84.00	138.58	204.71
113	6	1	2	4	1	4	o	0.5171	78.00	131.76	210.75
114	6	2	3	1	2	4	o	0.6754	91.00	128.43	199.30
115	6	3	2	2	2	4	o	0.8419	104.00	145.32	204.23
116	6	1	3	2	2	4	o	0.6004	82.00	125.70	201.31
117	6	3	1	3	2	4	o	0.6960	103.00	150.14	207.48
118	6	2	2	3	2	4	o	0.7976	89.00	142.38	207.28
119	6	1	1	4	2	4	o	0.4757	78.00	135.85	208.74
120	6	1	3	1	3	4	o	0.5101	89.00	126.42	211.64
121	6	3	1	2	3	4	o	0.6867	108.00	150.14	210.54
122	6	2	2	2	3	4	o	0.7870	94.00	142.38	210.34
123	6	1	1	3	3	4	o	0.4819	81.00	134.92	201.51
124	6	1	1	1	4	4	o	0.3898	87.00	126.30	202.22
125	6	2	2	1	1	5	o	0.5594	102.00	121.86	208.34
126	6	2	1	2	1	5	o	0.5370	99.00	125.96	204.31
127	6	1	2	2	1	5	o	0.4972	93.00	119.13	210.35
128	6	2	1	1	2	5	o	0.5146	102.00	125.96	206.32
129	6	1	1	2	2	5	o	0.4575	93.00	123.23	208.34

n_f= 600 n_n=129 6 0.0100 0.0465

Results for example 4c:

List of noninferior and efficient point

p= 116.0000 c= 190.0000 w= 218.0000

1	1	5	2	1	1	1	*	0.5707	114.00	134.40	163.92
2	1	2	5	1	1	1	o	0.5365	72.00	110.20	206.38
3	2	4	4	2	1	1	*	0.7206	105.00	140.29	206.72

4	2	1	5	2	1	1	o	0.4769	63.00	107.47	208.40
5	3	5	1	3	1	1	*	0.5662	116.00	148.78	181.27
6	3	2	4	4	1	1	o	0.6676	69.00	134.15	214.30
7	3	3	2	5	1	1	o	0.6862	74.00	147.49	210.49
8	3	1	3	5	1	1	o	0.4893	52.00	127.86	207.57
9	3	5	1	1	2	1	o	0.5251	114.00	138.50	161.91
10	3	4	4	1	2	1	o	0.6905	108.00	140.29	208.74
11	3	1	5	1	2	1	o	0.4570	66.00	107.47	210.41
12	4	4	3	3	2	1	*	0.8462	102.00	153.71	203.51
13	4	3	4	3	2	1	o	0.8320	88.00	145.71	217.07
14	4	4	2	4	2	1	o	0.8111	99.00	159.05	207.68
15	4	3	3	4	2	1	o	0.8275	81.00	150.51	208.03
16	4	3	1	5	2	1	o	0.6313	74.00	151.58	208.48
17	4	2	2	5	2	1	o	0.7235	60.00	143.83	208.28
18	4	3	4	1	3	1	o	0.6841	90.00	136.15	206.03
19	4	4	3	2	3	1	o	0.8349	107.00	153.71	206.56
20	4	1	4	3	3	1	o	0.5761	66.00	130.49	211.10
21	4	4	1	4	3	1	o	0.6616	103.00	163.87	213.99
22	4	3	2	4	3	1	o	0.8036	81.00	154.91	204.98
23	4	2	3	4	3	1	o	0.7735	71.00	147.57	214.14
24	4	2	1	5	3	1	o	0.5901	64.00	148.65	214.58
25	4	1	4	1	4	1	o	0.4659	72.00	121.87	211.81
26	4	4	2	2	4	1	o	0.7974	111.00	159.05	215.26
27	4	3	3	2	4	1	o	0.8136	93.00	150.51	215.60
28	4	3	2	3	4	1	o	0.8007	88.00	154.91	209.50
29	4	1	3	3	4	1	o	0.5710	66.00	135.29	206.58
30	4	2	1	4	4	1	o	0.5911	69.00	147.45	204.71
31	4	1	2	4	4	1	o	0.5473	63.00	140.63	210.75
32	4	3	1	1	5	1	o	0.5168	92.00	142.03	208.54
33	4	2	2	1	5	1	o	0.5923	78.00	134.27	208.34
34	4	2	1	2	5	1	o	0.5686	75.00	138.37	204.31
35	4	1	2	2	5	1	o	0.5265	69.00	131.54	210.35
36	4	1	1	3	5	1	o	0.4352	68.00	136.36	213.60
37	4	4	4	1	1	2	o	0.6605	111.00	137.56	208.74
38	4	1	5	1	1	2	o	0.4372	69.00	104.74	210.41
39	4	4	3	3	1	2	o	0.8094	105.00	150.98	203.51
40	4	3	4	3	1	2	o	0.7959	91.00	142.98	217.07
41	4	4	2	4	1	2	o	0.7758	102.00	156.32	207.68
42	4	3	3	4	1	2	o	0.7916	84.00	147.78	208.03
43	4	3	1	5	1	2	o	0.6039	77.00	148.85	208.48
44	4	2	2	5	1	2	o	0.6920	63.00	141.10	208.28

45	5	4	3	2	2	2	!	0.9008	106.00	150.26	198.24
46	5	3	4	2	2	2	o	0.8857	92.00	142.25	211.81
47	5	2	4	3	2	2	o	0.8391	77.00	139.32	214.86
48	5	4	1	4	2	2	o	0.7138	102.00	160.41	205.67
49	5	3	2	4	2	2	o	0.8670	80.00	151.46	196.65
50	5	2	3	4	2	2	o	0.8346	70.00	144.12	205.81
51	5	2	1	5	2	2	o	0.6367	63.00	145.19	206.26
52	5	1	2	5	2	2	o	0.5895	57.00	138.37	212.30
53	5	4	3	1	3	2	o	0.7653	113.00	150.98	208.57
54	5	2	4	2	3	2	o	0.8279	82.00	139.32	217.91
55	6	4	2	3	3	2	*	0.9038	111.00	164.94	216.56
56	7	3	3	3	3	2	!	0.9222	93.00	156.40	216.91
57	7	3	1	4	3	2	o	0.7071	84.00	156.28	202.96
58	7	2	2	4	3	2	o	0.8104	70.00	148.52	202.76
59	7	3	3	1	4	2	o	0.7458	99.00	147.78	217.62
60	7	4	1	2	4	2	o	0.7017	114.00	160.41	213.24
61	7	3	2	2	4	2	o	0.8524	92.00	151.46	204.23
62	7	2	3	2	4	2	o	0.8205	82.00	144.12	213.39
63	7	3	1	3	4	2	o	0.7046	91.00	156.28	207.48
64	7	2	2	3	4	2	o	0.8075	77.00	148.52	207.28
65	7	1	1	4	4	2	o	0.4816	66.00	141.99	208.74
66	7	2	1	1	5	2	o	0.5212	81.00	135.64	206.32
67	7	1	2	1	5	2	o	0.4826	75.00	128.82	212.36
68	7	1	1	2	5	2	o	0.4633	72.00	132.91	208.34
69	7	3	4	1	1	3	o	0.6477	98.00	130.49	206.03
70	7	4	3	2	1	3	o	0.7904	115.00	148.05	206.56
71	7	1	4	3	1	3	o	0.5454	74.00	124.82	211.10
72	7	4	1	4	1	3	o	0.6263	111.00	158.20	213.99
73	7	3	2	4	1	3	o	0.7607	89.00	149.25	204.98
74	7	2	3	4	1	3	o	0.7323	79.00	141.91	214.14
75	7	2	1	5	1	3	o	0.5587	72.00	142.98	214.58
76	7	2	4	2	2	3	o	0.8194	87.00	136.38	217.91
77	7	4	2	3	2	3	o	0.8945	116.00	162.00	216.56
78	7	3	3	3	2	3	o	0.9127	98.00	153.47	216.91
79	7	3	1	4	2	3	o	0.6999	89.00	153.34	202.96
80	7	2	2	4	2	3	o	0.8021	75.00	145.59	202.76
81	7	3	3	1	3	3	o	0.7504	100.00	143.91	205.87
82	7	1	4	1	3	3	o	0.5157	82.00	124.82	216.17
83	7	4	1	2	3	3	o	0.7061	115.00	156.55	201.49
84	7	2	3	2	3	3	o	0.8256	83.00	140.25	201.64
85	7	3	2	3	3	3	o	0.8862	98.00	157.87	213.86

86	7	1	3	3	3	3	o	0.6320	76.00	138.25	210.94
87	7	2	1	4	3	3	o	0.6542	79.00	150.41	209.07
88	7	1	2	4	3	3	o	0.6057	73.00	143.58	215.11
89	7	3	2	1	4	3	o	0.7168	104.00	149.25	214.56
90	7	1	3	1	4	3	o	0.5111	82.00	129.63	211.64
91	7	3	1	2	4	3	o	0.6881	101.00	153.34	210.54
92	7	2	2	2	4	3	o	0.7886	87.00	145.59	210.34
93	7	2	1	3	4	3	o	0.6519	86.00	150.41	213.59
94	7	1	4	1	1	4	o	0.4402	87.00	113.00	211.81
95	7	3	3	2	1	4	o	0.7687	108.00	141.64	215.60
96	7	3	2	3	1	4	o	0.7565	103.00	146.05	209.50
97	7	1	3	3	1	4	o	0.5395	81.00	126.42	206.58
98	7	2	1	4	1	4	o	0.5585	84.00	138.58	204.71
99	7	1	2	4	1	4	o	0.5171	78.00	131.76	210.75
100	7	3	3	1	2	4	o	0.7367	111.00	141.64	217.62
101	7	3	2	2	2	4	o	0.8419	104.00	145.32	204.23
102	7	2	3	2	2	4	o	0.8105	94.00	137.98	213.39
103	7	3	1	3	2	4	o	0.6960	103.00	150.14	207.48
104	7	2	2	3	2	4	o	0.7976	89.00	142.38	207.28
105	7	1	1	4	2	4	o	0.4757	78.00	135.85	208.74
106	7	3	2	1	3	4	o	0.7153	111.00	146.05	214.56
107	7	1	3	1	3	4	o	0.5101	89.00	126.42	211.64
108	7	3	1	2	3	4	o	0.6867	108.00	150.14	210.54
109	7	2	2	2	3	4	o	0.7870	94.00	142.38	210.34
110	7	2	1	3	3	4	o	0.6506	93.00	147.20	213.59
111	7	2	1	1	4	4	o	0.5262	99.00	138.58	214.30
112	7	1	1	2	4	4	o	0.4677	90.00	135.85	216.31
113	7	3	1	1	1	5	o	0.4881	116.00	129.62	208.54
114	7	2	2	1	1	5	o	0.5594	102.00	121.86	208.34
115	7	2	1	2	1	5	o	0.5370	99.00	125.96	204.31
116	7	1	2	2	1	5	o	0.4972	93.00	119.13	210.35
117	7	1	1	3	1	5	o	0.4111	92.00	123.95	213.60
118	7	2	1	1	2	5	o	0.5146	102.00	125.96	206.32
119	7	1	2	1	2	5	o	0.4765	96.00	119.13	212.36
120	7	1	1	2	2	5	o	0.4575	93.00	123.23	208.34
n_f=	642	n_n=120			7			0.0109	0.0583		

Results for example 4d:

List of noninferior and efficient point

p= 116.0000 c= 145.0000 w= 236.0000

1	1	5	2	1	1	1	*	0.5707	114.00	134.40	163.92
2	1	3	5	1	1	1	o	0.5852	92.00	123.41	224.70
3	1	5	1	2	1	1	o	0.5479	111.00	138.50	159.89
4	2	4	4	2	1	1	*	0.7206	105.00	140.29	206.72
5	2	2	5	2	1	1	o	0.6438	75.00	119.75	220.48
6	3	4	3	3	1	1	*	0.7358	96.00	144.16	187.40
7	3	1	5	3	1	1	o	0.4928	68.00	117.74	229.77
8	3	4	1	4	1	1	o	0.5642	87.00	144.04	173.45
9	3	3	3	4	1	1	o	0.7196	75.00	140.96	191.92
10	3	3	1	5	1	1	o	0.5490	68.00	142.03	192.37
11	3	2	3	5	1	1	o	0.6606	64.00	140.15	219.65
12	3	1	1	6	1	1	o	0.3729	47.00	130.47	228.04
13	3	5	1	1	2	1	o	0.5251	114.00	138.50	161.91
14	3	4	4	1	2	1	o	0.6905	108.00	140.29	208.74
15	3	2	5	1	2	1	o	0.6170	78.00	119.75	222.49
16	4	4	3	2	2	1	*	0.8189	97.00	143.43	182.13
17	4	1	5	2	2	1	o	0.5484	69.00	117.02	224.50
18	4	4	1	3	2	1	o	0.6447	86.00	142.38	157.90
19	5	3	3	3	2	1	*	0.8222	74.00	139.30	176.37
20	5	3	2	4	2	1	o	0.7881	71.00	144.64	180.55
21	6	2	4	4	2	1	*	0.7677	75.00	143.70	230.41
22	6	2	2	5	2	1	o	0.7235	60.00	143.83	208.28
23	6	1	3	5	2	1	o	0.5627	58.00	137.42	223.68
24	6	4	3	1	3	1	o	0.6958	104.00	144.16	192.47
25	6	1	5	1	3	1	o	0.4660	76.00	117.74	234.84
26	6	4	1	2	3	1	o	0.6361	91.00	142.38	160.96
27	6	3	3	2	3	1	o	0.8113	79.00	139.30	179.43
28	6	3	2	3	3	1	o	0.7984	74.00	143.71	173.32
29	7	2	4	3	3	1	*	0.7777	78.00	142.77	223.18
30	7	2	2	4	3	1	o	0.7367	61.00	141.70	186.65
31	7	1	2	5	3	1	o	0.5464	58.00	141.82	220.62
32	7	4	1	1	4	1	o	0.5316	102.00	144.04	183.04
33	7	3	3	1	4	1	o	0.6780	90.00	140.96	201.51
34	7	2	4	1	4	1	o	0.6290	84.00	134.15	223.89
35	7	3	2	2	4	1	o	0.7749	83.00	144.64	188.12
36	7	1	4	2	4	1	o	0.5591	75.00	131.42	225.90
37	7	2	2	3	4	1	o	0.7341	68.00	141.70	191.18
38	7	1	2	4	4	1	o	0.5473	63.00	140.63	210.75
39	7	3	1	1	5	1	o	0.5168	92.00	142.03	208.54
40	7	2	3	1	5	1	o	0.6219	88.00	140.15	235.82
41	7	2	2	2	5	1	o	0.7107	81.00	143.83	222.43

42	7	1	2	3	5	1	o	0.5440	74.00	141.82	231.72
43	7	4	4	1	1	2	o	0.6605	111.00	137.56	208.74
44	7	2	5	1	1	2	o	0.5902	81.00	117.02	222.49
45	7	4	3	2	1	2	o	0.7833	100.00	140.71	182.13
46	7	1	5	2	1	2	o	0.5246	72.00	114.29	224.50
47	7	4	1	3	1	2	o	0.6167	89.00	139.65	157.90
48	8	3	4	3	1	2	*	0.7959	91.00	142.98	217.07
49	8	3	2	4	1	2	o	0.7539	74.00	141.91	180.55
50	8	2	4	4	1	2	o	0.7344	78.00	140.97	230.41
51	8	2	2	5	1	2	o	0.6920	63.00	141.10	208.28
52	8	1	3	5	1	2	o	0.5383	61.00	134.69	223.68
53	8	4	3	1	2	2	o	0.7506	103.00	140.71	184.15
54	8	1	5	1	2	2	o	0.5027	75.00	114.29	226.52
55	9	4	2	2	2	2	*	0.8579	96.00	144.39	170.76
56	10	3	4	2	2	2	!	0.8857	92.00	142.25	211.81
57	10	3	2	3	2	2	o	0.8614	73.00	140.25	165.00
58	10	2	4	3	2	2	o	0.8391	77.00	139.32	214.86
59	10	2	3	4	2	2	o	0.8346	70.00	144.12	205.81
60	10	1	4	4	2	2	o	0.6256	72.00	138.24	234.43
61	10	1	2	5	2	2	o	0.5895	57.00	138.37	212.30
62	10	4	1	1	3	2	o	0.5831	97.00	139.65	162.97
63	10	3	4	1	3	2	o	0.7525	99.00	142.98	222.14
64	10	3	2	2	3	2	o	0.8499	78.00	140.25	168.05
65	10	2	4	2	3	2	o	0.8279	82.00	139.32	217.91
66	11	2	3	3	3	2	!	0.8454	73.00	143.19	198.59
67	11	1	4	3	3	2	o	0.6337	75.00	137.31	227.21
68	11	2	1	4	3	2	o	0.6483	64.00	143.07	184.64
69	11	1	3	4	3	2	o	0.6303	68.00	142.11	218.16
70	11	1	1	5	3	2	o	0.4808	61.00	143.19	218.61
71	11	3	2	1	4	2	o	0.7103	89.00	141.91	190.14
72	11	1	4	1	4	2	o	0.5125	81.00	128.69	227.92
73	11	2	3	2	4	2	o	0.8205	82.00	144.12	213.39
74	11	2	1	3	4	2	o	0.6460	71.00	143.07	189.16
75	11	1	3	3	4	2	o	0.6281	75.00	142.11	222.68
76	11	1	1	4	4	2	o	0.4816	66.00	141.99	208.74
77	11	2	2	1	5	2	o	0.6515	87.00	141.10	224.44
78	11	1	2	2	5	2	o	0.5791	78.00	138.37	226.46
79	11	1	1	3	5	2	o	0.4787	77.00	143.19	229.71
80	11	4	3	1	1	3	o	0.6587	112.00	138.49	192.47
81	11	1	5	1	1	3	o	0.4411	84.00	112.08	234.84
82	11	4	2	2	1	3	o	0.7528	105.00	142.17	179.08

83	11	3	4	2	1	3	o	0.7772	101.00	140.04	220.13
84	12	3	3	3	1	3	*	0.7936	92.00	143.91	200.80
85	12	2	4	3	1	3	o	0.7363	86.00	137.10	223.18
86	12	3	1	4	1	3	o	0.6086	83.00	143.79	186.85
87	12	2	3	4	1	3	o	0.7323	79.00	141.91	214.14
88	12	2	1	5	1	3	o	0.5587	72.00	142.98	214.58
89	12	1	2	5	1	3	o	0.5173	66.00	136.16	220.62
90	12	4	2	1	2	3	o	0.7214	108.00	142.17	181.09
91	12	3	4	1	2	3	o	0.7448	104.00	140.04	222.14
92	13	3	3	2	2	3	!	0.8832	93.00	143.19	195.53
93	13	2	4	2	2	3	o	0.8194	87.00	136.38	217.91
94	13	3	1	3	2	3	o	0.6954	82.00	142.13	171.31
95	13	2	3	3	2	3	o	0.8368	78.00	140.25	198.59
96	13	1	4	3	2	3	o	0.6272	80.00	134.37	227.21
97	13	2	1	4	2	3	o	0.6416	69.00	140.13	184.64
98	13	1	3	4	2	3	o	0.6238	73.00	139.18	218.16
99	13	1	1	5	2	3	o	0.4759	66.00	140.25	218.61
100	13	3	3	1	3	3	o	0.7504	100.00	143.91	205.87
101	13	2	4	1	3	3	o	0.6962	94.00	137.10	228.25
102	13	3	1	2	3	3	o	0.6861	87.00	142.13	174.36
103	13	2	3	2	3	3	o	0.8256	83.00	140.25	201.64
104	13	1	4	2	3	3	o	0.6188	85.00	134.37	230.26
105	13	2	2	3	3	3	o	0.8125	78.00	144.66	195.53
106	13	1	3	3	3	3	o	0.6320	76.00	138.25	210.94
107	13	1	2	4	3	3	o	0.6057	73.00	143.58	215.11
108	13	3	1	1	4	3	o	0.5734	98.00	143.79	196.44
109	13	2	3	1	4	3	o	0.6900	94.00	141.91	223.72
110	13	2	1	2	4	3	o	0.6308	81.00	140.13	192.22
111	13	1	3	2	4	3	o	0.6133	85.00	139.18	225.74
112	13	1	2	3	4	3	o	0.6036	80.00	143.58	219.63
113	13	2	1	1	5	3	o	0.5259	96.00	142.98	230.75
114	13	1	1	2	5	3	o	0.4675	87.00	140.25	232.76
115	13	2	4	1	1	4	o	0.5943	99.00	125.28	223.89
116	13	3	3	2	1	4	o	0.7687	108.00	141.64	215.60
117	13	1	4	2	1	4	o	0.5283	90.00	122.55	225.90
118	13	3	1	3	1	4	o	0.6052	97.00	140.59	191.38
119	13	2	3	3	1	4	o	0.7283	93.00	138.71	218.66
120	13	2	2	4	1	4	o	0.6981	90.00	144.04	222.83
121	13	3	3	1	2	4	o	0.7367	111.00	141.64	217.62
122	13	1	4	1	2	4	o	0.5063	93.00	122.55	227.92
123	13	3	1	2	2	4	o	0.6736	98.00	139.86	186.11

124	13	2	3	2	2	4	o	0.8105	94.00	137.98	213.39
125	13	2	2	3	2	4	o	0.7976	89.00	142.38	207.28
126	13	1	3	3	2	4	o	0.6204	87.00	135.98	222.68
127	13	1	2	4	2	4	o	0.5947	84.00	141.31	226.86
128	13	3	1	1	3	4	o	0.5723	105.00	140.59	196.44
129	13	2	3	1	3	4	o	0.6886	101.00	138.71	223.72
130	13	2	2	2	3	4	o	0.7870	94.00	142.38	210.34
131	13	1	3	2	3	4	o	0.6121	92.00	135.98	225.74
132	13	1	2	3	3	4	o	0.6024	87.00	140.38	219.63
133	13	2	2	1	4	4	o	0.6577	105.00	144.04	232.42
134	13	1	2	2	4	4	o	0.5846	96.00	141.31	234.43
135	13	3	1	1	1	5	o	0.4881	116.00	129.62	208.54
136	13	2	3	1	1	5	o	0.5874	112.00	127.73	235.82
137	13	2	2	2	1	5	o	0.6713	105.00	131.41	222.43
138	13	2	1	3	1	5	o	0.5549	104.00	136.23	225.68
139	13	1	2	3	1	5	o	0.5138	98.00	129.41	231.72
140	13	2	2	1	2	5	o	0.6433	108.00	131.41	224.44
141	13	2	1	2	2	5	o	0.6176	105.00	135.51	220.42
142	13	1	2	2	2	5	o	0.5718	99.00	128.68	226.46
143	13	1	1	3	2	5	o	0.4727	98.00	133.50	229.71
144	13	2	1	1	3	5	o	0.5247	112.00	136.23	230.75
145	13	1	1	2	3	5	o	0.4664	103.00	133.50	232.76
n_f=	629	n_n=145		13				0.0207	0.0897		

Results for example 4e:

List of noninferior and efficient point

p= 90.00000 c= 195.0000 w= 256.0000

1	1	4	3	1	1	1	*	0.5934	88.00	124.33	151.93
2	2	4	2	3	1	1	*	0.7008	86.00	138.29	159.92
3	2	2	5	3	1	1	o	0.6653	80.00	130.03	241.85
4	2	4	1	4	1	1	o	0.5642	87.00	144.04	173.45
5	3	3	4	4	1	1	*	0.7282	89.00	147.36	232.62
6	3	1	4	5	1	1	o	0.4951	66.00	134.27	248.27
7	3	2	1	6	1	1	o	0.5034	59.00	142.75	240.12
8	3	1	2	6	1	1	o	0.4661	53.00	135.93	246.16
9	4	4	2	2	2	1	*	0.7799	87.00	137.56	154.65
10	4	4	1	3	2	1	o	0.6447	86.00	142.38	157.90
11	5	3	4	3	2	1	*	0.8320	88.00	145.71	217.07
12	5	1	5	3	2	1	o	0.5667	74.00	127.30	245.88
13	6	3	3	5	2	1	*	0.8286	90.00	162.91	254.08
14	6	1	1	6	2	1	o	0.4289	53.00	140.03	244.14

15	6	4	1	1	3	1	o	0.5301	88.00	132.83	146.86
16	6	3	4	1	3	1	o	0.6841	90.00	136.15	206.03
17	6	2	5	1	3	1	o	0.6291	88.00	130.03	246.92
18	6	1	5	2	3	1	o	0.5592	79.00	127.30	248.93
19	7	3	3	3	3	1	*	0.8383	84.00	149.58	200.80
20	8	2	4	4	3	1	*	0.7828	85.00	153.98	254.83
21	8	3	2	5	3	1	o	0.8046	90.00	167.32	251.03
22	8	1	3	5	3	1	o	0.5737	68.00	147.70	248.11
23	8	3	3	1	4	1	o	0.6780	90.00	140.96	201.51
24	8	2	4	2	4	1	o	0.7548	87.00	143.70	237.98
25	8	3	2	3	4	1	o	0.8007	88.00	154.91	209.50
26	8	1	4	3	4	1	o	0.5778	80.00	141.70	247.28
27	8	3	1	4	4	1	o	0.6447	89.00	160.67	223.03
28	8	2	3	4	4	1	o	0.7758	85.00	158.78	250.31
29	8	2	1	5	4	1	o	0.5918	78.00	159.86	250.76
30	8	2	3	1	5	1	o	0.6219	88.00	140.15	235.82
31	8	2	2	3	5	1	o	0.7344	86.00	154.10	243.80
32	8	1	1	4	5	1	o	0.4380	75.00	147.57	245.26
33	8	1	1	1	6	1	o	0.3510	82.00	130.47	253.64
34	8	4	2	2	1	2	o	0.7460	90.00	134.83	154.65
35	8	3	4	2	1	2	o	0.7702	86.00	132.70	195.70
36	8	4	1	3	1	2	o	0.6167	89.00	139.65	157.90
37	8	1	5	3	1	2	o	0.5421	77.00	124.57	245.88
38	8	1	1	6	1	2	o	0.4102	56.00	137.30	244.14
39	8	3	4	1	2	2	o	0.7381	89.00	132.70	197.71
40	8	4	1	2	2	2	o	0.6863	90.00	138.93	152.64
41	9	2	5	2	2	2	*	0.8144	90.00	136.13	252.69
42	10	3	3	4	2	2	!	0.9103	90.00	157.33	224.14
43	10	2	4	4	2	2	o	0.8445	84.00	150.52	246.51
44	10	3	2	5	2	2	o	0.8681	89.00	163.86	242.70
45	10	2	3	5	2	2	o	0.8356	79.00	156.52	251.86
46	10	1	5	1	3	2	o	0.5126	85.00	124.57	250.95
47	10	3	3	2	3	2	o	0.8924	88.00	146.12	195.53
48	11	2	4	3	3	2	!	0.8555	87.00	149.59	239.29
49	11	3	2	4	3	2	o	0.8839	90.00	161.74	221.08
50	11	2	3	4	3	2	o	0.8509	80.00	154.40	230.24
51	11	2	2	5	3	2	o	0.8114	79.00	160.93	248.81
52	11	3	2	1	4	2	o	0.7103	89.00	141.91	190.14
53	11	3	1	2	4	2	o	0.6819	86.00	146.00	186.11
54	11	1	4	2	4	2	o	0.6150	84.00	138.24	242.01
55	11	2	3	3	4	2	o	0.8479	87.00	154.40	234.77

56	11	2	2	4	4	2	o	0.8127	84.00	159.73	238.94
57	11	1	3	4	4	2	o	0.6321	82.00	153.32	254.34
58	11	1	1	5	4	2	o	0.4822	75.00	154.40	254.79
59	11	2	2	2	5	2	o	0.7818	90.00	150.65	238.54
60	11	1	3	2	5	2	o	0.6081	88.00	144.24	253.94
61	11	2	1	3	5	2	o	0.6463	89.00	155.47	241.79
62	11	1	2	3	5	2	o	0.5984	83.00	148.65	247.83
63	11	3	3	2	1	3	o	0.7680	87.00	133.64	179.43
64	11	1	5	2	1	3	o	0.5294	87.00	121.63	248.93
65	11	2	4	3	1	3	o	0.7363	86.00	137.10	223.18
66	11	3	2	4	1	3	o	0.7607	89.00	149.25	204.98
67	11	1	4	4	1	3	o	0.5489	81.00	136.03	242.75
68	11	1	3	5	1	3	o	0.5431	76.00	142.03	248.11
69	11	3	3	1	2	3	o	0.7360	90.00	133.64	181.44
70	11	1	5	1	2	3	o	0.5073	90.00	121.63	250.95
71	11	2	4	2	2	3	o	0.8194	87.00	136.38	217.91
72	11	3	2	3	2	3	o	0.8692	88.00	147.59	189.43
73	11	3	1	4	2	3	o	0.6999	89.00	153.34	202.96
74	11	2	3	4	2	3	o	0.8422	85.00	151.46	230.24
75	11	2	2	5	2	3	o	0.8031	84.00	157.99	248.81
76	11	3	2	1	3	3	o	0.7147	90.00	138.04	178.39
77	11	3	1	2	3	3	o	0.6861	87.00	142.13	174.36
78	12	2	3	3	3	3	!	0.8531	88.00	150.53	223.02
79	12	1	4	3	3	3	o	0.6395	90.00	144.65	251.64
80	12	2	2	4	3	3	o	0.8177	85.00	155.87	227.19
81	12	1	3	4	3	3	o	0.6360	83.00	149.46	242.59
82	12	2	1	5	3	3	o	0.6550	88.00	162.81	255.12
83	12	2	2	2	4	3	o	0.7886	87.00	145.59	210.34
84	12	2	1	3	4	3	o	0.6519	86.00	150.41	213.59
85	12	1	3	3	4	3	o	0.6338	90.00	149.46	247.11
86	12	1	2	4	4	3	o	0.6075	87.00	154.79	251.29
87	12	1	2	1	5	3	o	0.4870	90.00	136.16	236.79
88	12	1	1	2	5	3	o	0.4675	87.00	140.25	232.76
89	12	3	1	1	1	4	o	0.4881	89.00	120.76	155.91
90	12	2	3	2	1	4	o	0.7048	88.00	128.43	197.28
91	12	1	4	2	1	4	o	0.5283	90.00	122.55	225.90
92	12	2	2	4	1	4	o	0.6981	90.00	144.04	222.83
93	12	1	3	4	1	4	o	0.5429	88.00	137.63	238.23
94	12	2	2	3	2	4	o	0.7976	89.00	142.38	207.28
95	12	1	3	3	2	4	o	0.6204	87.00	135.98	222.68
96	12	2	1	4	2	4	o	0.6422	90.00	148.14	220.82

97	12	1	2	4	2	4	o	0.5947	84.00	141.31	226.86
98	12	1	1	5	2	4	o	0.4763	87.00	148.26	254.79
99	12	1	3	1	3	4	o	0.5101	89.00	126.42	211.64
100	12	2	1	2	3	4	o	0.6296	88.00	136.93	192.22
101	12	1	2	3	3	4	o	0.6024	87.00	140.38	219.63
102	12	1	1	4	3	4	o	0.4850	88.00	146.13	233.17
103	12	1	1	2	4	4	o	0.4677	90.00	135.85	216.31
104	12	1	2	1	1	5	o	0.4144	90.00	109.58	196.26
105	12	1	1	2	1	5	o	0.3978	87.00	113.67	192.23
106	12	1	1	1	2	5	o	0.3812	90.00	113.67	194.24
n_f=	665	n_n=106		12				0.0180	0.1132		

8.2 Programme codes

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>



int x[1000],numelements, constraints,flag[100],xu[100],xopt[1000];
float reliability[1000],minweights[100],weights[100][100],resource[100];
float r1[1000],r2[1000],sys=0,sys1=0;
void cls()
{ system("cls");
}

main()
{
  FILE *output;
  int i=1,j,k,which,checkinf,checkmmt,t=2,v=-1,kk,temp,checkn,tt=0;
  char out[30];

/*  void upperbound(int,int,int,int*,int*,float*,float*);*/
  cls();


  for(i=0;i<100;i++)
     {flag[i]=-5;
     }
  printf("Program for calculating the optimal reliability of series-parallel s
  \n");

  re1:
  printf("\nHow many components are there? ");
  scanf("%d",&numelements);

  if( (numelements != (int)numelements) || (numelements<=0) )
    {printf("The number of components should be a postive integer!");
     goto re1;}
```

```
  for(i=0;i<numelements;i++)
     {x[i]=1;
     }

  i=0;

  while(i<numelements)
     {

      printf("\nplease enter the reliability for component %d :", i+1);
      scanf("%f",&reliability[i]);

      if(reliability[i]>1|| reliability[i]<=0)
{printf("\nReliabilities for components must be in the range of (0,1]!");
 printf("\nPlease enter again!");
 i--;
}
      i++;
     }


   for(i=0;i<numelements;i++)
      {r1[i]=log(1-reliability[i]);
       r2[i]=log(reliability[i]);
      }


   printf("\nHow many constraints are there ?");
   scanf("%d",&constraints);

/*   weights=calloc(constraints,numelements*sizeof(float));*/
   i=0;

   while(i<constraints)
{ cls();
  printf("\nWeights entering for constraint no. %d\n\n",i+1);
  k=0;
  while(k<numelements)
       { printf("\nPlease enter the weight for element %d :",k+1);
```

```
 scanf("%f",&weights[i][k]);
k++;
}

 printf("\nPlease enter the resource available :");
 scanf("%f",&resource[i]);



 printf("\nPlease indicate whether it is 1) less than or equal to");
 printf("\n                              or 2) less than            : ");

 scanf("%d",&flag[i]);

 i++;
}




    printf("\nPlease enter the output file name: ");
    scanf("%s",&out);


    output=fopen(out,"w+");


    fprintf(output,"\n               List of Maximal monotone points");
    fprintf(output,"\nNo.          X          Maximal monotone     R(X) ");
    fprintf(output,"\n-------------------------------------------------------

    for(which=0;which<numelements;which++)
{upperbound(which);
 xu[which]=x[which];
 x[which]=1;
}
```

```
    x[0]=xu[0];                /*start from (xu,x,x,x) */
    for(i=1;i<numelements;i++)
       {x[i]=1;}


    findmin();



step1:
      checkn=0;

      checkinf=0;
      checkinf=noninf();

      checkmmt=0;
      checkmmt=mmt();


      if(checkinf&&checkmmt)
{for(i=0;i<numelements;i++)
    {xopt[i]=x[i];
    }

goto step7;
}



step2:

      x[1]=x[1]++;


step3:

      temp=x[0];
      upperbound(0);

      if(x[0]>0)
```

```
{goto step6;}
      x[0]=temp;


step4:
       v=v+1;
       if(v>numelements-3)
 {goto stop;}


step5:
       kk=t+v;
       x[kk]=x[kk]+1;

       if(x[kk]>xu[kk])
 {goto step4;}

       for(i=1;i<=kk-1;i++)
  {x[i]=1;}
  v=-1;
  goto step3;


step6:
       checkinf=0;
       checkinf=noninf();

       checkmmt=0;
       checkmmt=mmt();

       checkn=1;
       checkn=test();


 if(checkinf==0 || checkmmt==0)
 {goto step2;
 }


step7:
```

```
      sys1=1;

      for(i=0;i<numelements;i++)
        { sys1=sys1* (1-pow((1-reliability[i]),x[i]) );
        }

      if(sys1>sys)
        {sys=sys1;

for(i=0;i<numelements;i++)
 {xopt[i]=x[i];

  }
       }

       {tt++;
fprintf(output,"\n%3d  (",tt);

for(i=0;i<numelements-1;i++)
   {fprintf(output,"%d,",x[i]);
   }

fprintf(output,"%d)",x[numelements-1]);

if(checkn==0){fprintf(output,"              *              ");}
if(checkn==1){fprintf(output,"              !              ");}

fprintf(output,"%1.5f",sys1);
       }


      goto step2;




stop:
      fprintf(output,"\n\ The optimal point is at   : (");

      for(i=0;i<numelements-1;i++)
 {fprintf(output,"%d,",xopt[i]);
```

```
 }

      fprintf(output,"%d)",xopt[numelements-1]);

      fprintf(output,"\n The optimal reliability is: %1.5f",sys);

      fclose(output);




}


mmt()
{int xtemp[1000],temp,i,j,k,l,count=0,flag1=1;
 float *cost,tempc[100];
 cost=malloc(constraints);


 for(i=0;i<numelements;i++)
    { xtemp[i]=x[i];}


 for(i=0;i<constraints;i++)
    {cost[i]=0;
     for (j=0;j<numelements;j++)
 {cost[i]=cost[i]+weights[i][j]*x[j];


 }
     tempc[i]=cost[i];
     }

 for(i=0;i<numelements;i++)
    {for(j=i+1;j<numelements;j++)
{
  while (xtemp[j]>xtemp[i])
{count=0;
```

```
 xtemp[i]++;

 xtemp[j]--;

 for(k=0;k<constraints;k++)
    {cost[k]=cost[k]+weights[k][i]-weights[k][j];

     if(flag[k]==1 && cost[k]<=resource[k])
       {count++;

       }

     if(flag[k]==2 && cost[k]<resource[k])
       {count++;

       }
    }

 if(count==constraints)
   {flag1=0;
    break;
   }


}


for(k=0;k<constraints;k++)
   {cost[k]=tempc[k];}

for(k=0;k<numelements;k++)
   { xtemp[k]=x[k];}


}

    }
```

```
  return(flag1);
}

findmin()
{ int i,j;

  for(i=0;i<constraints;i++)
     {
      minweights[i]=10000;

      for(j=0;j<numelements;j++)
 {if(weights[i][j]<minweights[i])
    {minweights[i]=weights[i][j];

     }

  }

     }

}



int noninf()
{int i,j,k,count=0,flag1=0;
 float cost,remain;


 for(i=0;i<constraints;i++)
    {cost=0;
     remain=1000;

     for(j=0;j<numelements;j++)
{cost=cost+weights[i][j]*x[j];
}

     remain=resource[i]-cost;

     if(remain<minweights[i])
```

```
        {flag1=1;}

      if(flag[i]==1&&remain <0)
        {flag1=2;}  /*infeasible*/

      if(flag[i]==2&&remain<=0)
        {flag1=2;}
     }


 return(flag1);

}


upperbound(int which)
{
  int i,j,k,okay=1;
  float* cost;

  cost=malloc(constraints);


  for(i=0;i<constraints;i++)
     {cost[i]=0;
     }




  while(okay==1)
       {  x[which]++;


  for(i=0;i<constraints;i++)
     {cost[i]=0;
      for(j=0;j<numelements;j++)

  { cost[i]=cost[i]+weights[i][j]*x[j];
```

```
  }

     }


 for(i=0;i<constraints;i++)
    {if (flag[i]==1&& cost[i]>resource[i])
{ okay=0;
re1:   if(x[which]>0)
     {x[which]--;

      for(k=0;k<constraints;k++)
 {cost[k]=cost[k]-weights[k][which];}


      for(k=0;k<constraints;k++)
 {if(cost[k]>resource[k])
     {goto re1;}
 }

     }

  break;
}

     if (flag[i]==2 && cost[i]>=resource[i])
{ okay=0;
re2:   if(x[which]>0)
    {x[which]--;

     for(k=0;k<constraints;k++)
{cost[k]=cost[k]-weights[k][which];}

     for(k=0;k<constraints;k++)
{if(cost[k]>=resource[k])
    {goto re2;}
}
```

```
      }

   break;}
     }
        }


}


test()
{ int i,j,k,diff,flag1,count=0,xtemp[1000];
  float a,b,c,n,*cost;
  flag1=1;
  n=20000;

  cost=malloc(constraints);

  for(i=0;i<numelements;i++)
     {xtemp[i]=x[i];
     }


  for(i=0;i<constraints;i++)
     {cost[i]=0;
      for(j=0;j<numelements;j++)
 { cost[i]=cost[i]+weights[i][j]*x[j];

 }
     }


  for(i=0;i<numelements;i++)
     {for (j=i+1;j<numelements;j++)
  {
   diff=x[i]-x[j];
   if(diff>=2)
     {a=r2[j];
      b=log(reliability[i]+(reliability[j]-reliability[i]) * pow((1-reliabilit
```

```
       c=x[j]*(r1[j]-r1[i]);
       n=(a-b+c)/r1[i];
       n++;

       if(diff>n)
{while(xtemp[i]>1)
   { xtemp[i]--;
     xtemp[j]++;


     for(k=0;k<constraints;k++)
       {cost[k]=cost[k]+weights[k][j]-weights[k][i];
       }


     for(k=0;k<constraints;k++)
{if(flag[k]==1 && cost[k]<=resource[k])
   {count++;}
 if(flag[k]==2 && cost[k]<resource[k])
   {count++;}
}


     if(count==constraints)
       {flag1=0;
return(flag1);
       }
   }

}



    }

  }

    }
```

```
return(flag1);

}
```

[20] [21] [22] [23] [24] [25] [26]

References

[1] D. Li, "Successive solution scheme for constrained redundancy optimization in reliability networks," *Working paper of the SEEM department, CUHK*, 1998.

[2] A. J. Federowicz and M. Mazumdar, "Use of geometric programming to maximize reliability achieved by redundancy," *Operations Research*, vol. 16, no. 5, pp. 948–954, 1968.

[3] S. M. Ross, "Introduction to probability models," *New York: Academic Press*, vol. 2nd edition, 1980.

[4] W. K. Frank A. Tillman, Ching-Lai Hwang, "Optimization of systems reliability," *Marcel Dekker, inc*, 1980.

[5] S. Tzafestas, "Optimization of system reliability: a survey of problems and techniques," *Int. J. Systems Sci.*, vol. 11, no. 4, pp. 455–486, 1980.

[6] M. K.B. and J. Sharma, "A new geometric programming formulation for a reliability problem," *International journal of quality control*, vol. 18, no. 3, pp. 497–503, 1973.

[7] J. Sharma and K. Venkateswaran, "A direct method for maximizing the system reliability," *IEEE transactions on Reliability*, vol. R-20, no. 4, pp. 256–259, 1971.

[8] K. B. Misra, "A simple approach for constrained redundancy optimization problems," *IEEE transactions on reliability*, vol. R-21, no. 1, pp. 30–34, 1972.

[9] D. H. Shi, "A new heuristic algorithm for constrained redundancy-optimization in complex systems," *IEEE Transactions On Reliability*, vol. R-36, pp. 621–623, Dec. 1987.

[10] X. W. Chen Huacheng, Shi Dinghua, "A new algorithm for network system reliability," *Microelectron. and rel.*, vol. 25, no. 1, pp. 35–40, 1985.

[11] J. D. Kettlle, "Least cost allocation of reliability investment," *Operations Research*, vol. 10, pp. 249–265, 1967.

[12] K. Misra and U. Sharma, "An efficient algorithm to solve integer-programming problem arising in system-reliability design," *IEEE Transactions On Reliability*, vol. 40, Apr. 1991.

[13] O. Berman and N. Ashrafi, "Optimization models for reliability of

modular software systems," *IEEE transactions on software engineering*, vol. 19, Nov. 1993.

[14] J. D. Musa, "A theory of software reliability and its application," *IEEE Transaction software engineering*, vol. SE-1, no. 3, pp. 312–327, 1975.

[15] H. Sarper, "No special schemes are needed for solving software reliability optimization models," *IEEE transactions on software engineering*, vol. 21, Aug. 1993.

[16] "Lingo optimization midelling language," *Chicago: LINDO systems inc.*, 1983.

[17] X. Sun and D. Li, "On characterization of optimal solution in constraint redundancy optimization problems," *Working paper of the SEEM department, CUHK*, 1998.

[18] R. Bellman and S. Dreyfus, "Dynamic programming and the reliability of multicomponent devices," *operations research*, no. 6, pp. 200–206, 1958.

[19] W. Winston, *Introduction to mathematical programming, Applications and Algorithms*, vol. 2nd edition. 1995.

[20] J. E. M. Brian Borchers, "A computational comparison of branch and

bound and outer approximation algorithms for 0-1 mixed integer nonlinear programs," *Computers ops. Res.*, vol. 24, no. 8, pp. 699–701, 1997.

[21] M. Sniedovich, *Dynamic Programming.* M. Dekker, 1992.

[22] D. M. Z. Peng Tian, Jian Ma, "Non-linear integer programming by darwin and boltzmann mixed strategy," *European journal of operation research*, pp. 224–235, 1998.

[23] D. P. Bertsekas, "Dynamic programming: deterministic and stochastic models," *Prentice hall*, pp. 1–46, 1987.

[24] C. Kilmister, *Lagrangian dynamics: an introduction for students.* London : logos P., 1967.

[25] w. W. H. David E. Fyffe and N. K. Lee, "System reliability allocation and a computational algorithm," *IEEE transactions on reliability*, vol. R-17, June 1968.

[26] F. S. Hillier and G. J. Lieberman, *Introduction to operations research.* Mcgraw-hill international editions, 1990.